Tensor Product Attention Is All You Need

Yifan Zhang[*,◇,1] **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, substantially shrinking the KV cache size at inference time. By factorizing these representations into contextual low-rank components and seamlessly integrating with Rotary Position Embedding (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 on language modeling tasks, we demonstrate that T6 surpasses or matches the performance of standard Transformer baselines, including Multi-Head Attention (MHA), Multi-Query Attention (MQA), Grouped-Query Attention (GQA), and Multi-Head Latent Attention (MLA) across various metrics, including perplexity and a range of established evaluation benchmarks. Notably, TPA's memory efficiency and computational efficiency at the decoding stage enable processing 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 [4, 11, 57, 5]. As these models evolve, their ability to process longer contexts becomes increasingly important for sophisticated applications such as document analysis, complex reasoning, and code completion. However, managing longer sequences during inference poses significant computational and memory challenges, particularly due to the storage of key-value (KV) caches [69, 33]. 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 [9] or token eviction strategies [69, 61, 41], though such methods risk discarding tokens that may later prove important. Other work proposes off-chip storage of key-value states [16], at the expense of increased I/O latency. Attention variants like Multi-Query Attention (MQA) [45] and Grouped-Query Attention

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

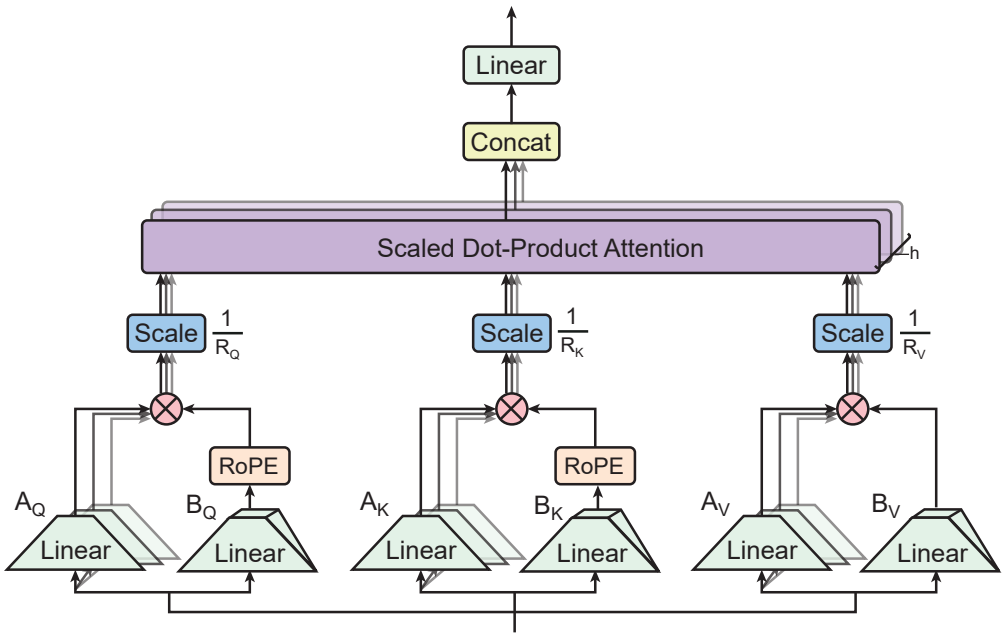


Figure 1: Tensor Product Attention (TPA) within the **T**ensor Produc**T** A**TT**en**T**ion **T**ransformer (T6). In each TPA layer, the input hidden state $\mathbf{x}_t$ is processed by linear layers to produce latent factor matrices for query (e.g., $\mathbf{A}_Q(\mathbf{x}_t), \mathbf{B}_Q(\mathbf{x}_t)$), key (e.g., $\mathbf{A}_K(\mathbf{x}_t), \mathbf{B}_K(\mathbf{x}_t)$), and value (e.g., $\mathbf{A}_V(\mathbf{x}_t), \mathbf{B}_V(\mathbf{x}_t)$). Rotary Position Embedding (RoPE) is applied to the $\mathbf{B}_Q(\mathbf{x}_t)$ and $\mathbf{B}_K(\mathbf{x}_t)$ factors. The query, key, and value tensors for each attention head are then formed by the tensor product of these factor matrices (e.g., $\mathbf{Q}_t = \frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top \mathbf{B}_Q(\mathbf{x}_t)$). Finally, the TPA output is computed using scaled dot-product attention, followed by a linear projection of the concatenated results from all heads.

(GQA) [2] 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 [19] effectively reduce fine-tuning memory, yet do not address the KV cache overhead that dominates inference at runtime. The recently introduced Multi-Head Latent Attention (MLA) in Deepseek-V2 [31] caches compressed key-value representations but encounters difficulties with efficient Rotary Position Embedding (RoPE) [51] integration, necessitating additional position-encoded parameters per head.

To overcome the limitations of existing approaches, we introduce Tensor Product Attention (TPA), illustrated in Figure 1. TPA is a novel attention mechanism that employs tensor factorizations for queries (Q), keys (K), and values (V). By dynamically factorizing *activations* rather than static weights (as in LoRA), TPA constructs low-rank, contextual representations. This approach substantially reduces KV cache memory usage while offering improved representational capacity. In practice, TPA can decrease memory overhead by an order of magnitude compared to standard Multi-Head Attention (MHA), alongside achieving lower pretraining validation loss (perplexity) and better downstream performance.

A key advantage of TPA is its native compatibility with rotary positional embeddings (RoPE) [51], enabling a straightforward drop-in replacement for multi-head attention (MHA) layers in modern LLM architectures such as LLaMA [57] and Gemma [55].

Our main contributions are summarized as follows:

1. We propose **Tensor Product Attention (TPA)**, a mechanism that factorizes $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ activations using *contextual* tensor decompositions. This achieves a substantial reduction in inference-time KV cache size relative to standard attention mechanisms [59], MHA, MQA, GQA, and MLA, while also improving performance. In addition, we analyze existing attention mechanisms and reveal that MHA, MQA, and GQA can be expressed as non-contextual variants of TPA.
2. We introduce the **T**ensor Produc**T** A**TT**en**T**ion **T**ransformer (T6), a new TPA-based model architecture for sequence modeling. In language modeling experiments, T6 consistently improves or matches validation perplexity and downstream evaluation performance, all while maintaining a reduced KV cache size.
3. We demonstrate that TPA integrates seamlessly with RoPE [51], facilitating its easy adoption in popular foundation model architectures like LLaMA and Gemma.
4. We develop **FlashTPA Decoding**, an efficient autoregressive inference algorithm for TPA. Our empirical results show that FlashTPA Decoding can be faster than optimized MHA, MQA, GQA, and MLA decoding methods, particularly for long sequences.

2 Background

In this section, we briefly review Scaled Dot-Product Attention, Multi-Head Attention [59] and introduce key notations. Other attention mechanisms like Multi-Query Attention (MQA) [45], Grouped Query Attention (GQA) [2], Multi-head Latent Attention (MLA) [31, 32], and Rotary Position Embedding (RoPE) [51] are further discussed in Appendix H.

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 projection matrices 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: $\mathbf{a} \otimes \mathbf{b} = \mathbf{C} \in \mathbb{R}^{m \times n}$, with $C_{ij} = a_i b_j$, where a_i is the i-th element of $\mathbf{a}$, b_j is the j-th element of $\mathbf{b}$, and C_{ij} is the (i, j)-th entry of $\mathbf{C}$. The vectorization of a matrix $\mathbf{C} \in \mathbb{R}^{m \times n}$, denoted $\text{vec}(\mathbf{C}) \in \mathbb{R}^{mn}$, stacks the columns of $\mathbf{C}$ into a single column vector. For example, if $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n]$ where $\mathbf{c}_j$ are columns, then $\text{vec}(\mathbf{C}) = [\mathbf{c}_1^\top, \mathbf{c}_2^\top, \ldots, \mathbf{c}_n^\top]^\top$.

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

2.1 Scaled Dot-Product Attention

Scaled dot-product attention [59] 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 values ($\mathbf{V}$). Formally, the attention output is:

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

where each of $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ is an $(n \times d_k)$ matrix for n tokens and key dimension d_k.

2.2 Multi-Head Attention (MHA)

Multi-Head Attention (MHA) [59] 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 from different representational subspaces. For each token embedding $\mathbf{x}_t \in \mathbb{R}^{d_{\text{model}}}$, MHA computes each head i as follows:

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

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

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

MHA enables the model to capture a rich set of dependencies by allowing each head to focus on different aspects of the input sequence. We also discuss how MHA relates to TPA in Appendix A. Further discussion on Transformers and attention mechanisms, KV cache optimization techniques, and low-rank factorization methods is available in Appendix G.

3 Tensor Product Attention

In this section, we provide a detailed description of our proposed *Tensor Product Attention* (TPA), which enables *contextual* low-rank factorization for queries, keys, and values. First, we explain how TPA factorizes these components, specifying tensor shapes. Next, we describe TPA's integration into the multi-head attention framework and its benefits for reducing KV cache memory consumption during inference. Finally, we demonstrate RoPE's seamless integration with TPA, including a pre-rotated variant for efficiency.

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. 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:

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

$$\mathbf{V}_t = \frac{1}{R_V}\sum_{r=1}^{R_V} \mathbf{a}_r^V(\mathbf{x}_t) \otimes \mathbf{b}_r^V(\mathbf{x}_t), \tag{3.1}$$

where $\mathbf{a}_r^Q(\mathbf{x}_t), \mathbf{a}_r^K(\mathbf{x}_t), \mathbf{a}_r^V(\mathbf{x}_t) \in \mathbb{R}^h$, $\mathbf{b}_r^Q(\mathbf{x}_t), \mathbf{b}_r^K(\mathbf{x}_t), \mathbf{b}_r^V(\mathbf{x}_t) \in \mathbb{R}^{d_h}$. Hence, for queries, each tensor product $\mathbf{a}_r^Q(\mathbf{x}_t) \otimes \mathbf{b}_r^Q(\mathbf{x}_t)\colon \mathbb{R}^h \times \mathbb{R}^{d_h} \to \mathbb{R}^{h\times d_h}$ (an outer product) contributes to the query slice $\mathbf{Q}_t \in \mathbb{R}^{h\times d_h}$. Analogous definitions apply to the 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:

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

where $\boldsymbol{W}_r^{a^Q} \in \mathbb{R}^{h\times d_{\text{model}}}$ and $\boldsymbol{W}_r^{b^Q} \in \mathbb{R}^{d_h\times d_{\text{model}}}$ are learnable weight matrices. Similar linear maps produce the factors for keys and values.

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

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

which are then reshaped into $\mathbf{A}_Q(\mathbf{x}_t) \in \mathbb{R}^{R_Q\times h}$ and $\mathbf{B}_Q(\mathbf{x}_t) \in \mathbb{R}^{R_Q\times d_h}$ (where each row of $\mathbf{A}_Q(\mathbf{x}_t)$ corresponds to an $\mathbf{a}_r^Q(\mathbf{x}_t)^\top$ and each row of $\mathbf{B}_Q(\mathbf{x}_t)$ to a $\mathbf{b}_r^Q(\mathbf{x}_t)^\top$). The query tensor for token t can then be expressed as:

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

This operation is equivalent to $\mathbf{Q}_t = \frac{1}{R_Q}\sum_{r=1}^{R_Q} \mathbf{a}_r^Q(\mathbf{x}_t)(\mathbf{b}_r^Q(\mathbf{x}_t))^\top$, where $\mathbf{a}_r^Q$ is the r-th column of $\mathbf{A}_Q(\mathbf{x}_t)^\top$ and $(\mathbf{b}_r^Q)^\top$ is the r-th row of $\mathbf{B}_Q(\mathbf{x}_t)$. Repeating for all tokens reconstitutes $\mathbf{Q} \in \mathbb{R}^{T\times h\times d_h}$. Similar procedures are 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 $i \in \{1, \ldots, h\}$:

$$\mathbf{head}_i = \text{Softmax}\Big(\tfrac{1}{\sqrt{d_h}}\,\mathbf{Q}_i\,(\mathbf{K}_i)^\top\Big)\,\mathbf{V}_i, \tag{3.2}$$

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 $\boldsymbol{W}^O \in \mathbb{R}^{(h\cdot d_h)\times d_{\text{model}}}$:

$$\text{TPA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}\big(\mathbf{head}_1, \ldots, \mathbf{head}_h\big)\boldsymbol{W}^O. \tag{3.3}$$

Parameter Initialization. We use Xavier initialization [14] for the factor weight matrices; details are in Appendix I.

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:

$$\mathbf{Q}_t \mapsto \widetilde{\mathbf{Q}}_t = \mathrm{RoPE}_t(\mathbf{Q}_t), \qquad \mathbf{K}_s \mapsto \widetilde{\mathbf{K}}_s = \mathrm{RoPE}_s(\mathbf{K}_s). \tag{3.4}$$

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

$$\widetilde{\mathbf{B}}_K(\mathbf{x}_t) \;\longleftarrow\; \mathrm{RoPE}_t\big(\mathbf{B}_K(\mathbf{x}_t)\big), \tag{3.5}$$

yielding a *pre-rotated* key representation:

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

Here, RoPE_t is applied to each row of $\mathbf{B}_K(\mathbf{x}_t)$ (i.e., to each $\mathbf{b}_{(r)}^K(\mathbf{x}_t)$ vector). Thus, each $\mathbf{K}_t$ is effectively rotated before caching. This removes the need for explicit rotation at decoding time, accelerating autoregressive inference. Depending on hardware and performance requirements, different RoPE integration strategies can be adopted for training and inference.

Theorem 3.1 (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:

$$\mathrm{RoPE}(\mathbf{Q}_t) = \frac{1}{R_Q} \mathbf{A}_Q(\mathbf{x}_t)^\top \widetilde{\mathbf{B}}_Q(\mathbf{x}_t), \tag{3.6}$$

where $\widetilde{\mathbf{B}}_Q(\mathbf{x}_t) = \mathrm{RoPE}_t\big(\mathbf{B}_Q(\mathbf{x}_t)\big)$ (RoPE applied row-wise to $\mathbf{B}_Q(\mathbf{x}_t)$). Furthermore, let $\mathbf{Q}_t$ and $\mathbf{K}_s$ be factorized by TPA. Let $\widetilde{\mathbf{Q}}_t = \mathrm{RoPE}_t(\mathbf{Q}_t)$ and $\widetilde{\mathbf{K}}_s = \mathrm{RoPE}_s(\mathbf{K}_s)$ be their RoPE-transformed versions. The relative positional encoding property of RoPE is preserved:

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

where RoPE_{t-s} denotes applying RoPE with relative position $t - s$. Focusing on individual heads i, if $\mathbf{q}_{t,i}$ and $\mathbf{k}_{s,i}$ are the i-th head vectors from $\mathbf{Q}_t$ and $\mathbf{K}_s$ respectively (as column vectors of dimension d_h), and $\widetilde{\mathbf{q}}_{t,i} = \mathrm{RoPE}(\mathbf{q}_{t,i}, t)$ and $\widetilde{\mathbf{k}}_{s,i} = \mathrm{RoPE}(\mathbf{k}_{s,i}, s)$, then the dot product for attention scores satisfies:

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

Theorem 3.1 indicates that TPA does not break RoPE's relative translational property. We prove Theorem 3.1 in Appendix F.1. In essence, RoPE_t applies a linear transformation (rotation matrix $\mathbf{R}_t$) to the d_h-dimensional space. Since $\mathbf{Q}_t = \frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top \mathbf{B}_Q(\mathbf{x}_t)$, applying RoPE yields $\mathbf{Q}_t\mathbf{R}_t = \frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top(\mathbf{B}_Q(\mathbf{x}_t)\mathbf{R}_t)$. Thus, RoPE effectively transforms $\mathbf{B}_Q(\mathbf{x}_t)$ to $\widetilde{\mathbf{B}}_Q(\mathbf{x}_t) = \mathbf{B}_Q(\mathbf{x}_t)\mathbf{R}_t$, where $\mathbf{R}_t$ acts on each row of $\mathbf{B}_Q(\mathbf{x}_t)$. The $\mathbf{A}_Q(\mathbf{x}_t)$ factor remains unchanged, 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 factor components. Specifically, for each past token t, we cache:

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

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

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 $R_K, R_V \ll h$ (e.g., rank 1 or 2) often yields substantial reduction of KV cache size. Table 1 provides a comparative overview of different attention mechanisms, including TPA and its variants, focusing on KV cache size per token and the number of parameters in an attention layer.

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 Appendix I. 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 Expressing MHA, MQA, and GQA as Non-contextual TPA

We demonstrate in Appendix A that several existing attention mechanisms, including MHA, MQA, and GQA, can be elegantly expressed as non-contextual variants of TPA by imposing specific constraints on the TPA factors.

3.5 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 $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{T \times h \times d_h}$

defined in Section 3.1, T6 utilize the overall architecture of LLaMA [57] while changing the self-attention block to our TPA-based version. The feed-forward network (FFN) adopts a SwiGLU layer, as in [46, 57].

The specific mechanisms for TPA-based query, key, and value factorization, as well as the attention computation within T6, directly follow the comprehensive descriptions provided in Section 3 (particularly Section 3.1 for factorization, and Equations 3.2 and 3.3 for attention computation). A brief recap of these steps within the T6 model context is provided in Appendix B.

Rotary Positional Embedding (RoPE). As discussed in Section 3.2, RoPE [51] 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 (3.5) and Theorem 3.1.

SwiGLU Feed-Forward Network. Following [46, 57], 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:

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

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

4 FlashTPA Decoding Algorithm

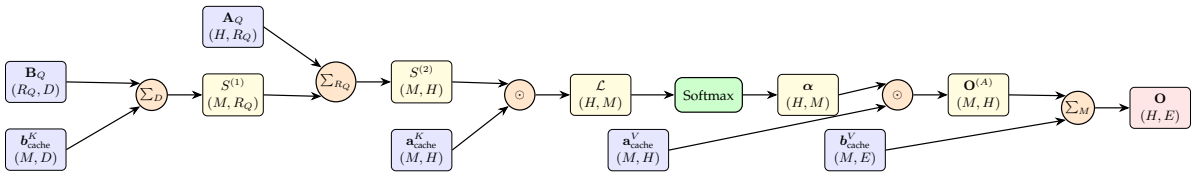


Figure 2: Data flow diagram for FlashTPA Decoding. Rectangles represent tensors (blue for inputs, yellow for intermediates, red for final output), circles with $\sum$ or $\odot$ denote Einstein summation contractions or element-wise products respectively, and the green rounded rectangle is the softmax operation. Shapes are shown for a single query ($N = 1$) interacting with M cached items. H is the number of heads, R_Q is the query rank, and D, E are respective feature dimensions for the $\mathbf{B}_Q/\boldsymbol{b}^K_{\text{cache}}$ and $\boldsymbol{b}^V_{\text{cache}}$ factors. Scaling factors in softmax are omitted for visual clarity.

For efficient autoregressive inference with Tensor Product Attention (TPA), we introduce FlashTPA Decoding. This algorithm is optimized for generating one token at a time by leveraging the factorized representation of queries, keys, and values. The core idea, illustrated in Figure 2, is to perform attention computations using a sequence of Einstein summations ("einsum") that operate directly on these factorized components. This avoids materializing the full query, key, and value tensors, which is particularly beneficial as the Key-Value (KV) cache grows with sequence length.

The detailed definitions of the input factorized components and the step-by-step pseudocode for FlashTPA Decoding are provided in Algorithm 1 (see Appendix D for details). An optimized Triton kernel implementation: Algorithm 3 is also outlined in Appendix D.

This sequence of factorized operations allows FlashTPA Decoding to compute the attention output efficiently. It minimizes memory allocations for large intermediate tensors and reduces the overall computational load compared to materializing full query, key, and value tensors, especially for long sequences. Consequently, TPA is not only memory-efficient due to its smaller KV cache footprint but can also be computationally efficient during inference. The experimental results for FlashTPA decoding time are presented in Section 5.2.

Algorithm 1 FlashTPA Decoding Algorithm

Require: $\mathbf{A}_Q$: Query A tensor $(B, 1, H, R_Q)$
Require: $\mathbf{B}_Q$: Query B tensor $(B, 1, R_Q, D)$
Require: $\mathbf{a}^K_{\text{cache}}$: Cached Key A tensor (B, M, H)
Require: $\boldsymbol{b}^K_{\text{cache}}$: Cached Key B tensor (B, M, D)
Require: $\mathbf{a}^V_{\text{cache}}$: Cached Value A tensor (B, M, H)
Require: $\boldsymbol{b}^V_{\text{cache}}$: Cached Value B tensor (B, M, E)
Require: $s_{\text{total}}, s_Q, s_K, s_V$: Scaling factors (with defaults $s_{\text{total}} \leftarrow 1/\sqrt{D}$, $s_Q \leftarrow 1/R_Q$)
Ensure: $\mathbf{O}$: Output tensor $(B, 1, H, E)$

▷ Step 1: Interaction between $\mathbf{B}_Q$ and $\boldsymbol{b}^K_{\text{cache}}$ components
$S^{(1)} \leftarrow \text{einsum}(\text{“bnrd, bmd} \rightarrow \text{bnmr”}, \mathbf{B}_Q, \boldsymbol{b}^K_{\text{cache}})$ ▷ Shape: $(B, 1, M, R_Q)$.
$S^{(1)}_{b,n,m,r} = \sum_d (\mathbf{B}_Q)_{b,n,r,d} \cdot (\boldsymbol{b}^K_{\text{cache}})_{b,m,d}$
▷ Step 2: Incorporate $\mathbf{A}_Q$ component
$S^{(2)} \leftarrow \text{einsum}(\text{“bnhr, bnmr} \rightarrow \text{bnmh”}, \mathbf{A}_Q, S^{(1)})$ ▷ Shape: $(B, 1, M, H)$.
$S^{(2)}_{b,n,m,h} = \sum_r (\mathbf{A}_Q)_{b,n,h,r} \cdot S^{(1)}_{b,n,m,r}$
▷ Step 3: Incorporate $\mathbf{a}^K_{\text{cache}}$ component to get full logits
$\mathcal{L} \leftarrow \text{einsum}(\text{“bnmh, bmh} \rightarrow \text{bhnm”}, S^{(2)}, \mathbf{a}^K_{\text{cache}})$ ▷ Shape: $(B, H, 1, M)$.
$\mathcal{L}_{b,h,n,m} = S^{(2)}_{b,n,m,h} \cdot (\mathbf{a}^K_{\text{cache}})_{b,m,h}$
▷ Step 4: Apply scaling and Softmax
$\boldsymbol{\alpha} \leftarrow \text{Softmax}_M(\mathcal{L} \cdot s_Q \cdot s_K \cdot s_{\text{total}})$ ▷ Shape: $(B, H, 1, M)$. Softmax over cache dimension M
▷ Step 5: Compute weighted sum with $\mathbf{a}^V_{\text{cache}}$ component
$\mathbf{O}^{(A)} \leftarrow \text{einsum}(\text{“bhnm, bmh} \rightarrow \text{bnmh”}, \boldsymbol{\alpha}, \mathbf{a}^V_{\text{cache}})$ ▷ Shape: $(B, 1, M, H)$.
$\mathbf{O}^{(A)}_{b,n,m,h} = \boldsymbol{\alpha}_{b,h,n,m} \cdot (\mathbf{a}^V_{\text{cache}})_{b,m,h}$
▷ Step 6: Incorporate $\boldsymbol{b}^V_{\text{cache}}$ component and final scaling
$\mathbf{O} \leftarrow \text{einsum}(\text{“bnmh, bme} \rightarrow \text{bnhe”}, \mathbf{O}^{(A)}, \boldsymbol{b}^V_{\text{cache}}) \cdot s_V$ ▷ Shape: $(B, 1, H, E)$. Final output
return $\mathbf{O}$

5 Experiments

5.1 Language Modeling Tasks

All experiments reported in this paper are implemented on the `nanoGPT` code base [23], using the FineWeb-Edu 100B dataset [36]. The dataset contains 100 billion tokens for training and 0.1 billion

tokens for validation. We compare T6 against the baseline Llama architecture [57] with SwiGLU activation [46] and RoPE embeddings [51], as well as Llama variants that replace Multi-Head Attention (MHA; [59]) with Multi-Query Attention (MQA; [45]), Grouped Query Attention (GQA; [2]), or Multi-head Latent Attention (MLA; [31]). 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 $4d^2_{\text{model}}$ parameters per attention layer. We train models at four scales: *small* (124M parameters), *medium* (353M), *large* (773M), and *XL* (1.5B). Details on architecture hyperparameters and training hardware are shown in Appendix J.1.

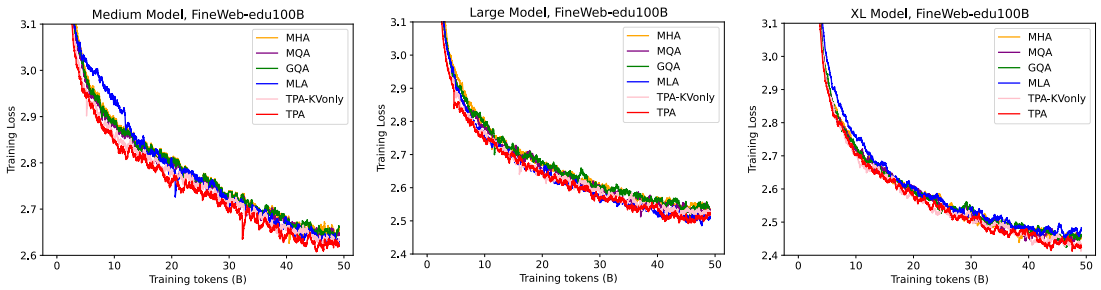
Training & Validation Curves. Figure 4 compares validation loss curves for the *medium* (353M), *large* (773M), and *XL* (1.5B) models on FineWeb-Edu-100B. Training loss curves are provided in Appendix Figure 3. Overall, **TPA** (red curves) and its simpler variant **TPA-KVonly** (pink curves) (see Appendix I) converge as fast as or faster than the baselines (MHA, MQA, GQA, MLA) while also achieving visibly lower final validation losses. For instance, in Figure 4(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) [31] (blue curves) generally trains more slowly and yields higher validation losses.

Validation Perplexity. Figure 9 (in the Appendix) 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 [62], BoolQ [12], HellaSwag [63], OBQA [38], PIQA [3], WinoGrande [42], and MMLU [17], using the `lm-evaluation-harness` codebase [13]. For ARC-E, ARC-C, HellaSwag, OBQA, PIQA, and SciQ, we report accuracy norm; for other tasks, we report standard accuracy. Due to the page limitation, we only display the zero-shot evaluation results of *medium* and *large* models here in Tables 2 and 3. Zero-shot evaluation of *small* and *XL* models are displayed in Tables 11 and 12 in the appendix. Moreover, we also present 2-shot evaluation results in Tables 13, 14, 15 and 16 in the appendix.

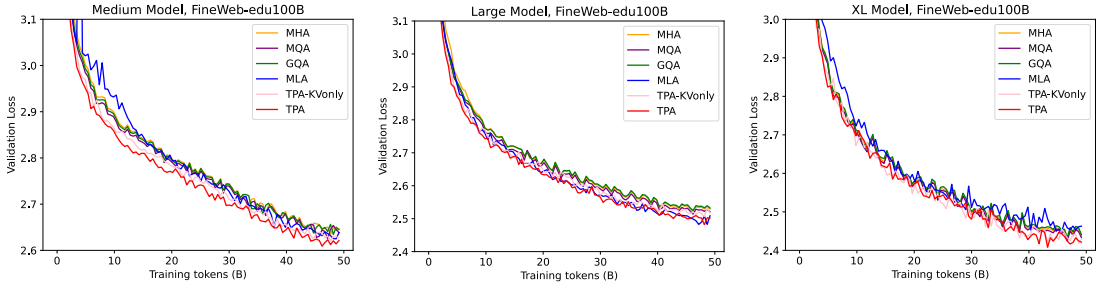
For the *medium*-size (353M) models (Table 2 for 0-shot and Table 14 in appendix for 2-shot), TPA generally ties or outperforms all competing methods, achieving, for example, an average of 51.41% in zero-shot mode versus MHA's 50.11%, MQA's 50.44%, and MLA's 50.13%. When given two-shot prompts, TPA again leads with 53.12% average accuracy. A similar trend appears for the *large*-size (773M) models (Table 3), where TPA-KVonly attains the highest average (53.52% zero-shot). For the *XL* size models (1.5B) (Table 12 in the appendix), TPA-KV only achieves the highest average (55.03% zero-shot).

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.



(a) Medium models (353M) (b) Large models (773M) (c) XL models (1.5B)

Figure 3: The training loss of medium-size (353M), large-size (773M) as well as XL-size (1.5B) models, with different attention mechanisms on the FineWeb-Edu 100B dataset.



(a) Medium models (353M) (b) Large models (773M) (c) XL models (1.5B)

Figure 4: The validation loss of medium-size (353M), large-size (773M) as well as XL-size (1.5B) models, with different attention mechanisms on the FineWeb-Edu 100B dataset.

Table 2: The evaluation results of medium 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.

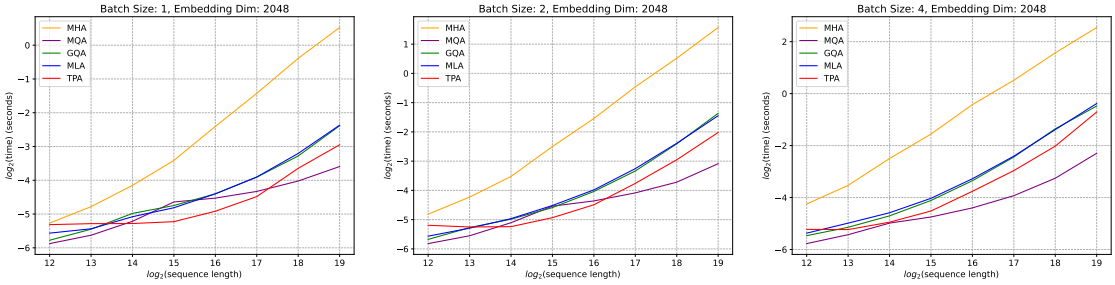
Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	**59.51**	29.52	**59.60**	45.68	34.20	68.82	53.43	23.33	76.90	50.11
MQA	57.62	**31.91**	59.45	45.69	35.40	69.31	53.51	**26.47**	74.60	50.44
GQA	58.67	31.48	58.29	45.45	35.20	68.50	**54.46**	24.58	76.50	50.35
MLA	56.65	29.52	57.83	46.05	34.60	69.42	52.80	24.62	79.70	50.13
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**

5.2 Experimental Results on FlashTPA Decoding

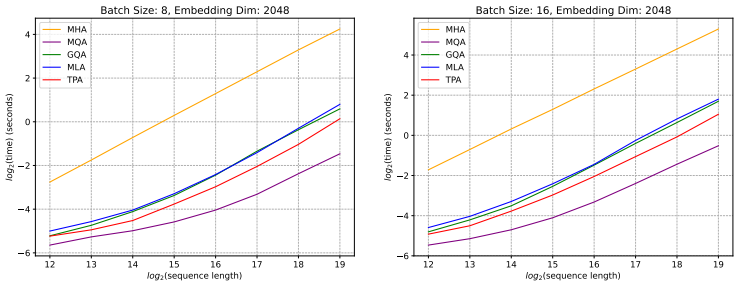
This section presents an evaluation of FlashTPA's decoding time in comparison to several other optimized attention mechanisms. We benchmark FlashTPA against FlashMHA [44], FlashGQA, FlashMQA, and FlashMLA [22]. It is important to note that our current FlashTPA implementation utilizes Triton [56]. While the compared methods are typically available as highly optimized CUDA kernels, these experiments provide initial insights into FlashTPA's potential. Development of a CUDA-based FlashTPA kernel is ongoing and is expected to yield further performance

Table 3: 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	**63.55**	32.85	60.95	**51.72**	**38.80**	70.51	55.01	24.55	**81.90**	53.32
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



(a) Batch Size=1 (b) Batch Size=2 (c) Batch Size=4



(d) Batch Size=8 (e) Batch Size=16

Figure 5: Decoding time comparison of different attention mechanisms with an embedding dimension of 2048 and $d_h = 64$. The y-axis represents $\log_2(\text{time})$ in seconds, and the x-axis represents $\log_2(\text{sequence length})$. Each subfigure corresponds to a different batch size.

improvements.

The evaluations were performed with batch sizes selected from $\{1, 2, 4, 8, 16\}$, model embedding dimensions (d_{model}) chosen from $\{1024, 2048, 3072\}$, and sequence lengths ranging from 2^{12} (4,096) to 2^{19} (524,288). For all experiments, the dimension per head (d_h) was fixed at 64. The ranks for TPA's factorized components (R_Q, R_K, R_V) were set to $(16, 1, 1)$, and for GQA configurations, the number of key-value head groups was 4.

The decoding time per token, measured as $\log_2(\text{time})$ in seconds, is plotted against $\log_2(\text{sequence length})$. Lower values on the y-axis indicate faster decoding times. Results are presented in Figure 5 for an embedding dimension of 2048 (corresponding to 32 attention heads). Additional results for embedding dimensions of 1024 (16 heads, Figure 8) and 3072 (48 heads,

Figure 7) are provided in Appendix D. Figure 5 depicts these speed comparisons for an embedding dimension of 2048. The results indicate that FlashTPA (blue line) is highly competitive and often outperforms other attention mechanisms, especially as the sequence length increases. A detailed breakdown of these comparisons across different batch sizes for the $d_{\text{model}} = 2048$ case (Figure 5) is provided in Appendix D.

These findings for an embedding dimension of 2048 underscore the computational efficiency of the FlashTPA decoding algorithm, particularly its favorable scaling with increasing sequence lengths.

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

[1] 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.

[2] 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, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 4895–4901. Association for Computational Linguistics, 2023.

[3] 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*, pages 7432–7439. AAAI Press, 2020.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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 *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, 2019.

[13] 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.

[14] 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*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[15] Insu Han, R Jayaram, A Karbasi, V Mirrokno, D Woodruff, and A Zandieh. Hyperattention: Long-context attention in near-linear time. In *International Conference on Learning Representations*. International Conference on Learning Representations, 2024.

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

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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

[23] Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

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

[25] 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)*, pages 155–172, 2024.

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

[27] 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.

[28] 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*, pages 23638–23647, 2024.

[29] 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*, 2024.

[30] 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*, 2024.

[31] 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*, 2024.

[32] 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*, 2024.

[33] 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*, 2024.

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

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

[36] Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024.

[37] 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*, pages 23610–23641. PMLR, 2023.

[38] 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 *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, 2018.

[39] 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, editors, *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*, pages 48–53. Association for Computational Linguistics, 2019.

[40] 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.

[41] 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.

[42] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740. Association for the Advancement of Artificial Intelligence (AAAI), 2020.

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

[44] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37:68658–68685, 2024.

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

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

[47] 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.

[48] 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*, 2023.

[49] 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.

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

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

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] Philippe Tillet, HT Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *ACM SIGPLAN International Workshop on Machine Learning and Programming Languages co-located with PLDI*. Association for Computing Machinery, 2019.

[57] 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.

[58] 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)*, pages 4344–4353, 2019.

[59] 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.

[60] 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*, pages 38087–38099. PMLR, 2023.

[61] 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.

[62] Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. 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)*, pages 2578–2589, 2019.

[63] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019.

[64] 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.

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

[66] Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Re. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. In *The Twelfth International Conference on Learning Representations*, 2024.

[67] 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, 2023.

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

[69] 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, 2023.

[70] 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*, pages 28631–28642, 2024.

Appendix

A Expressing MHA, MQA, and GQA as Non-contextual TPA

This appendix section details how standard Multi-Head Attention (MHA), Multi-Query Attention (MQA), and Grouped-Query Attention (GQA) can be expressed as special, non-contextual variants of Tensor Product Attention (TPA).

A.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 $\mathbf{x}_t$. More precisely, if we set $R_Q = R_K = R_V = h$ (the number of heads) and define the factors appropriately, MHA emerges. To match MHA with TPA, let $R_Q = R_K = R_V = h$. Focusing on $\mathbf{Q}_t$:

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

$$\mathbf{a}_i^Q = h\mathbf{e}_i \in \mathbb{R}^h, \tag{A.1}$$

where $\mathbf{e}_i \in \mathbb{R}^h$ is the i-th standard basis vector, so that $\mathbf{e}_i \otimes \cdot$ corresponds to the i-th head of $\mathbf{Q}_t$.

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

$$\mathbf{b}_i^Q(\mathbf{x}_t) = (\boldsymbol{W}_i^Q)^\top \mathbf{x}_t \in \mathbb{R}^{d_h}, \tag{A.2}$$

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

Substituting Equations (A.1) and (A.2) into the TPA formulation for $\mathbf{Q}_t$ (Equation 3.1 from the main paper, with scaling $1/R_Q = 1/h$ absorbed into $\mathbf{a}_i^Q$ for this example by setting $\mathbf{a}_i^Q = \mathbf{e}_i$ and adjusting the sum, or keeping the $1/h$ prefactor and setting $\mathbf{a}_i^Q = h\mathbf{e}_i$ as originally written, then adjusting the sum from $\sum_{r=1}^{R_Q}$ to $\sum_{i=1}^{h}$ and scaling by $1/h$): If we use $\mathbf{a}_i^Q = \mathbf{e}_i$ and sum up h such terms:

$$\mathbf{Q}_t = \sum_{i=1}^{h} \left[\mathbf{e}_i \otimes \left((\boldsymbol{W}_i^Q)^\top \mathbf{x}_t \right) \right] \in \mathbb{R}^{h \times d_h}. \tag{A.3}$$

Each term $\mathbf{e}_i \otimes \left((\boldsymbol{W}_i^Q)^\top \mathbf{x}_t \right)$ in Equation (A.3) is a matrix where only the i-th row is non-zero and equals $\left((\boldsymbol{W}_i^Q)^\top \mathbf{x}_t \right)^\top$. Summing these terms reconstitutes the usual MHA form of $\mathbf{Q}_t$. Analogous constructions hold for $\mathbf{K}_t$ and $\mathbf{V}_t$ using their respective projection matrices $\boldsymbol{W}_i^K, \boldsymbol{W}_i^V$. Thus, *MHA can be interpreted as a non-contextual, rank-h variant of TPA where the head-dimension factors are fixed basis vectors and the token-dimension factors are standard linear projections of the input*.

TPA with Non-contextual A factors. 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 the token-dimension factors $\mathbf{b}_r^Q(\mathbf{x}_t), \mathbf{b}_r^K(\mathbf{x}_t), \mathbf{b}_r^V(\mathbf{x}_t)$ to remain context-dependent. Then, for keys (using the original $1/R_K$ scaling):

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

and similarly for queries and values. This version, referred to as TPA (Non-contextual A), reduces per-token computations for the A factors and can be effective when head-dimension relationships are relatively stable across tokens. The KV cache for this variant would store $\mathbf{b}_r^K(\mathbf{x}_t)$ and $\mathbf{b}_r^V(\mathbf{x}_t)$ for each token, while $\mathbf{a}_r^K$ and $\mathbf{a}_r^V$ are fixed.

A.2 MQA and GQA as Non-contextual TPA

Multi-Query Attention (MQA) [45] and Grouped Query Attention (GQA) [2] (detailed in Appendix H.1 and H.2) also emerge naturally from TPA by restricting the head-dimension factors $(\mathbf{a}^K, \mathbf{a}^V)$ to be non-contextual *and* low-rank:

- **MQA as Rank-1 TPA (for K and V).** In MQA, all heads share a *single* set of keys and values. This corresponds to TPA with $R_K = R_V = 1$, where the head-dimension factors for keys and values are non-contextual vectors of all ones (or any constant non-zero vector, as it's a shared projection). For example:

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

  where $\mathbf{1}_h \in \mathbb{R}^h$ is a vector of ones. This forces every head to use the same key and value vectors $\mathbf{b}^K(\mathbf{x}_t)$ and $\mathbf{b}^V(\mathbf{x}_t)$ (which are effectively $\mathbf{X}\boldsymbol{W}^K_{\text{shared}}$ and $\mathbf{X}\boldsymbol{W}^V_{\text{shared}}$ from MQA's definition). Each head i would still use its distinct query $\mathbf{Q}_{t,i}$, which in TPA could be modeled with $R_Q = h$ and $\mathbf{a}_i^Q = \mathbf{e}_i$ as in the MHA case if queries are not factorized, or a different TPA structure for queries.

- **GQA as Grouped Rank-1 TPA (for K and V within groups).** GQA partitions h heads into G groups, with each group sharing keys and values. In TPA form, this means for keys and values, the rank R_K and R_V would be G. For each group $j \in [G]$, there would be a non-contextual head-dimension factor $\mathbf{a}_j^K \in \mathbb{R}^h$ (and similarly $\mathbf{a}_j^V$) that acts as a "mask" or selector for the heads in that group (e.g., $\mathbf{a}_j^K$ has ones for heads in group j and zeros elsewhere). Then:

$$\mathbf{K}_t = \frac{1}{G}\sum_{j=1}^{G} \mathbf{a}_j^K \otimes \mathbf{b}_j^K(\mathbf{x}_t), \quad \mathbf{V}_t = \frac{1}{G}\sum_{j=1}^{G} \mathbf{a}_j^V \otimes \mathbf{b}_j^V(\mathbf{x}_t),$$

  where $\mathbf{b}_j^K(\mathbf{x}_t)$ is the shared key vector for group j. Varying G from 1 (MQA) to h (MHA, if $\mathbf{b}_j^K$ are distinct per original head) shows GQA as an interpolation.

Hence, by constraining TPA's head-dimension factors $\mathbf{a}^K, \mathbf{a}^V$ to be specific constant vectors (or masks) and choosing appropriate ranks, these popular attention variants can be recovered as special cases of TPA with non-contextual A factors for keys and values. This underscores the flexibility and unifying nature of the TPA framework.

B T6 Model Architecture Details

This section provides a recap of how Tensor Product Attention (TPA) components are integrated within the **T**ensor Produc**T** A**TT**en**T**ion **T**ransformer (T6) architecture, building upon the detailed exposition in Section 3.

TPA QKV Factorization Recap. As defined in Section 3.1, for each token's hidden-state vector $\mathbf{x}_t \in \mathbb{R}^{d_{\text{model}}}$, TPA projects the input into three tensors $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{T \times h \times d_h}$. The slices for the

t-th token, $\mathbf{Q}_t,\ \mathbf{K}_t,\ \mathbf{V}_t \in \mathbb{R}^{h\times d_h}$, are formed by sums of tensor products. 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, for queries, with learnable weight matrices $\boldsymbol{W}_r^{a^Q} \in \mathbb{R}^{h\times d_{\text{model}}}$ and $\boldsymbol{W}_r^{b^Q} \in \mathbb{R}^{d_h\times d_{\text{model}}}$, these factors are:

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

In practice, these are often computed by merging ranks into a single linear layer followed by reshaping, as detailed in Section 3.1. The factorization for keys ($\mathbf{K}_t$) and values ($\mathbf{V}_t$) follows the same pattern using their respective ranks R_K, R_V and projection matrices.

Attention Step and Output Projection Recap. Once the factorized $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors are obtained for all tokens (with RoPE applied to the appropriate factors of $\mathbf{Q}$ and $\mathbf{K}$ as per Section 3.2), the attention output for each head $i \in \{1, \ldots, h\}$ is computed using the scaled dot-product attention formula (Equation 3.2 from the main paper):

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

Finally, the outputs from all h heads are concatenated and projected back to the model dimension using an output weight matrix $\boldsymbol{W}^O$, as shown in Equation 3.3 from the main paper:

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

This constitutes the output of the TPA sub-layer in a T6 block.

C Toward Faster Computation Without Materializing Q, K and V

In TPA, the query, key, and value tensors ($\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t$) are constructed as sums of R_Q, R_K, R_V rank-one tensors, respectively, as detailed in Equation 3.1. These sums are typically normalized by scaling factors $s_Q = 1/R_Q$, $s_K = 1/R_K$, and $s_V = 1/R_V$. This section explores the computational advantages of performing attention calculations directly on these factorized representations, including their scaling factors, without explicitly materializing the full $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors. This approach can potentially reduce floating-point operations (FLOPs) and memory bandwidth.

C.1 Single-Head Factorization Setup Without Materializing Q and K

Consider a single head i. Each query vector $\mathbf{Q}_t^{(i)} \in \mathbb{R}^{d_h}$ is factorized (with rank R_Q and scale $s_Q = 1/R_Q$):

$$\mathbf{Q}_t^{(i)} \;=\; s_Q \sum_{r=1}^{R_Q} a_{q,i}^{(r)}(\mathbf{x}_t)\,\mathbf{b}_q^{(r)}(\mathbf{x}_t),$$

and each key vector $\mathbf{K}_\tau^{(i)} \in \mathbb{R}^{d_h}$ is factorized (with rank R_K and scale $s_K = 1/R_K$):

$$\mathbf{K}_\tau^{(i)} \;=\; s_K \sum_{s=1}^{R_K} a_{k,i}^{(s)}(\mathbf{x}_\tau)\,\mathbf{b}_k^{(s)}(\mathbf{x}_\tau).$$

Their dot-product for tokens t, τ is

$$\left[\mathbf{Q}^{(i)}\,(\mathbf{K}^{(i)})^{\top}\right]_{t,\tau} \;=\; s_Q s_K \sum_{r=1}^{R_Q}\sum_{s=1}^{R_K} a_{q,i}^{(r)}(\mathbf{x}_t)\, a_{k,i}^{(s)}(\mathbf{x}_\tau)\, \big\langle \mathbf{b}_q^{(r)}(\mathbf{x}_t), \mathbf{b}_k^{(s)}(\mathbf{x}_\tau)\big\rangle. \tag{C.1}$$

Similarly, the value vector $\mathbf{V}_\tau^{(i)} \in \mathbb{R}^{d_h}$ is factorized (with rank R_V and scale $s_V = 1/R_V$):

$$\mathbf{V}_\tau^{(i)} \;=\; s_V \sum_{u=1}^{R_V} a_{v,i}^{(u)}(\mathbf{x}_\tau)\, \mathbf{b}_v^{(u)}(\mathbf{x}_\tau).$$

C.2 Multi-Head Case

For multi-head attention with h heads, one repeats the factorization across all heads. The $\mathbf{b}_q^{(r)}, \mathbf{b}_k^{(s)}, \mathbf{b}_v^{(u)}$ vectors are shared across heads. The scaling factors s_Q, s_K, s_V are applied per head as part of the definition.

C.3 Complexity Analysis

We compare the cost of standard multi-head attention versus TPA under two scenarios:

1. **Naïve:** Materialize $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ from factors (including their scales), then perform the usual batched GEMM.
2. **Specialized:** Attempt to compute $\mathbf{Q}\,\mathbf{K}^{\top}$ and the final attention output directly from the rank-(R_Q, R_K, R_V) factors and their scales, without explicitly forming full $\mathbf{Q}, \mathbf{K}, \mathbf{V}$.

Standard Multi-Head Attention. For batch size B and sequence length T:

- *Projection cost:* $\mathcal{O}\big(B\,T\,d_{\text{model}}^2\big)$ or $\mathcal{O}\big(B\,T\,d_{\text{model}}\,d_h\big)$.
- *Dot-product:* $\mathbf{Q}\,(\mathbf{K})^{\top} \in \mathbb{R}^{(B\,h)\times T\times T}$ costs $\mathcal{O}\big(B\,T^2\,d_{\text{model}}\big)$.

For large T, the $\mathcal{O}(B\,T^2\,d_{\text{model}})$ term dominates.

TPA: Naïve Implementation.

- *Constructing factors:* $\mathcal{O}\big(B\,T\,d_{\text{model}} \times (R_Q(h+d_h) + R_K(h+d_h) + R_V(h+d_h))\big)$.
- *Materializing* $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ *(including scaling):* $\mathcal{O}\big(B\,T\,(R_Q\,h\,d_h + R_K\,h\,d_h + R_V\,h\,d_h)\big)$. The scaling is a per-element multiplication.
- *Dot-product* $\mathbf{Q}\,(\mathbf{K})^{\top}$ *and multiply by* $\mathbf{V}$*:* $\mathcal{O}\big(B\,T^2\,d_{\text{model}}\big)$ for scores, and $\mathcal{O}\big(B\,T^2\,d_{\text{model}}\big)$ for value aggregation.

Typically $R_Q, R_K, R_V \ll h, d_h$, so the overhead of constructing factors and materializing is influenced by these ranks. The key benefit of TPA is in reducing KV cache size and potentially computation if specialized kernels are used.

TPA: Specialized Implementation. If we bypass explicitly forming $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, the computation involves the sums over ranks and the scaling factors. Below, we detail its complexity.

Algorithm 2 Specialized TPA Computation (Without Materializing $\mathbf{Q}, \mathbf{K}, \mathbf{V}$)

Require: Factorized Query components: $\mathbf{A}_Q(T, H, R_Q)$, $\mathbf{B}_Q(T, R_Q, d_h)$
Require: Factorized Key components: $\mathbf{A}_K(T, H, R_K)$, $\mathbf{B}_K(T, R_K, d_h)$
Require: Factorized Value components: $\mathbf{A}_V(T, H, R_V)$, $\mathbf{B}_V(T, R_V, d_v)$ (Note: d_v is value head dim, often $d_v = d_h$)
Require: Sequence length T, Number of heads H, Ranks R_Q, R_K, R_V, Head feature dimension d_h
Require: Scaling factors: $s_Q = 1/R_Q, s_K = 1/R_K, s_V = 1/R_V$ (or other pre-defined scales)
Ensure: Output tensor $\mathbf{O}(T, H, d_v)$

// Phase 1: Attention Score Computation

▷ Step 1.1: Compute shared $\mathbf{B}$-factor dot products P
$P \leftarrow \text{einsum}(\text{"tqrd, tkrd} \rightarrow \text{tqtkrqrk"}, \mathbf{B}_Q, \mathbf{B}_K)$
▷ Indices: t_q=query token, t_k=key token, r_q=query rank, r_k=key rank, d=feature dim
▷ Step 1.2: Combine P with $\mathbf{A}_Q, \mathbf{A}_K$ and apply s_Q, s_K to form scaled logits $\mathcal{L}'$
$\mathcal{L}_{\text{unscaled}} \leftarrow \text{einsum}(\text{"tqhrq, tkhrk, tqtkrqrk} \rightarrow \text{tqtkh"}, \mathbf{A}_Q, \mathbf{A}_K, P)$
$\mathcal{L}' \leftarrow s_Q \cdot s_K \cdot \mathcal{L}_{\text{unscaled}}$
▷ Indices: h=head
▷ Step 1.3: Apply attention scaling ($1/\sqrt{d_h}$) and Softmax to get probabilities $\boldsymbol{\alpha}$
$\mathcal{L}''_{\text{scaled}} \leftarrow \mathcal{L}'/\sqrt{d_h}$
$\boldsymbol{\alpha} \leftarrow \text{Softmax}_{T_k}(\mathcal{L}''_{\text{scaled}})$ ▷ Softmax over key tokens T_k for each query T_q and head H
▷ Output $\boldsymbol{\alpha}(t_q, t_k, h)$

// Phase 2: Value Aggregation

▷ Step 2.1: Conceptually weight $\mathbf{A}_V$ by $\boldsymbol{\alpha}$ to form $W_{\mathbf{A}_V}$
$W_{\mathbf{A}_V} \leftarrow \text{einsum}(\text{"tqtkh, tkhrv} \rightarrow \text{tqtkhrv"}, \boldsymbol{\alpha}, \mathbf{A}_V)$
▷ Indices: r_v=value rank
▷ Step 2.2: Aggregate $W_{\mathbf{A}_V}$ with $\mathbf{B}_V$ and apply s_V to get final output $\mathbf{O}$
$\mathbf{O}_{\text{unscaled}} \leftarrow \text{einsum}(\text{"tqtkhrv, tkrvd} \rightarrow \text{tqhd"}, W_{\mathbf{A}_V}, \mathbf{B}_V)$ ▷ d is value feature dim d_v
$\mathbf{O} \leftarrow s_V \cdot \mathbf{O}_{\text{unscaled}}$
return $\mathbf{O}$

C.4 Complexity Analysis for the Specialized Implementation

Complexity for a Single Query Token (Autoregressive Decoding). The dot product $\mathbf{Q}_t^{(i)} \cdot \mathbf{K}_\tau^{(i)}$ from Eq. (C.1) includes the $s_Q s_K$ scaling. For each pair (r, s), we pay:

1. $\mathcal{O}(1)$ for multiplying $a_{q,i}^{(r)}(\mathbf{x}_t)\, a_{k,i}^{(s)}(\mathbf{x}_\tau)$,
2. $\mathcal{O}(d_h)$ for the dot product $\langle \mathbf{b}_q^{(r)}(\mathbf{x}_t), \mathbf{b}_k^{(s)}(\mathbf{x}_\tau) \rangle$.

The $s_Q s_K$ scaling is a single multiplication after the sums. Since (r, s) runs over $R_Q \times R_K$, each token-pair (t, τ) costs roughly $\mathcal{O}\Big(R_Q\, R_K\, \big(1 + d_h\big)\Big) \approx \mathcal{O}(R_Q\, R_K\, d_h)$. For a single query token t and M cached key tokens $\tau = 1 \dots M$, this is $\mathcal{O}(M\, R_Q\, R_K\, d_h)$ for a single head to compute all unscaled dot products. The scaling by $s_Q s_K$ is $\mathcal{O}(M)$ for the score vector.

Multi-Head and Batches (Full Sequence Processing - reusing b-Dot Products as in Algorithm 2).

The b-vectors can be shared across heads. Let T be the sequence length.

1. **b-Dot-Product Stage (P):**
   Compute $P(t_q, \tau_k, r_q, r_k) = \langle \mathbf{b}_q^{(r_q)}(\mathbf{x}_{t_q}), \mathbf{b}_k^{(r_k)}(\mathbf{x}_{\tau_k}) \rangle$. Cost: $\mathcal{O}\big(T^2\, R_Q R_K\, d_h\big)$. This is shared across heads.
2. **Per-Head Score Computation:**
   For each head h, compute $\mathcal{L}_{t_q,\tau_k,h} = s_Q s_K \sum_{r_q,r_k} a_{q,h}^{(r_q)}(\mathbf{x}_{t_q})\, a_{k,h}^{(r_k)}(\mathbf{x}_{\tau_k})\, P(t_q, \tau_k, r_q, r_k)$.
   Cost: $\mathcal{O}\big(H\, T^2\, R_Q R_K\big)$.

Putting these together for batch size B, the total cost for scores is $\mathcal{O}\big(B\, T^2\, R_Q R_K\, d_h\big) + \mathcal{O}\big(B\, H\, T^2\, R_Q R_K\big) = \mathcal{O}\Big(B\, T^2\, R_Q R_K \big(d_h + H\big)\Big)$. This is for computing the scaled logits before softmax. The standard attention dot-product step is $\mathcal{O}\big(B\, T^2\, H\, d_h\big)$. For the specialized TPA to reduce flops in score computation: $R_Q R_K\, (d_h + H) < H\, d_h$. This implies $R_Q R_K < H/(1 + H/d_h)$. If $H \approx d_h$, then $R_Q R_K \approx H/2$.

C.5 Toward Faster Computation Without Materializing Q, K, V

We extend the idea to also avoid explicitly forming $\mathbf{V}$. After obtaining scaled logits $\mathcal{L}'_{t_q,\tau_k} = s_Q s_K (\mathbf{Q}_{t_q} \mathbf{K}_{\tau_k}^\top)$ (per head), we apply $\alpha_{t_q,\tau_k} = \mathrm{softmax}_{\tau_k}\big(\frac{1}{\sqrt{d_h}}\, \mathcal{L}'_{t_q,\tau_k}\big)$. The final attention output at token t_q (single head) using $\mathbf{V}_{\tau_k} = s_V \sum_{u=1}^{R_V} a_v^{(u)}(\mathbf{x}_{\tau_k})\, \mathbf{b}_v^{(u)}(\mathbf{x}_{\tau_k})$ is:

$$\mathrm{head}(t_q) \;=\; \sum_{\tau_k=1}^{T} \alpha_{t_q,\tau_k}\, \mathbf{V}_{\tau_k} \;=\; s_V \sum_{\tau_k=1}^{T} \alpha_{t_q,\tau_k} \sum_{u=1}^{R_V} a_v^{(u)}(\mathbf{x}_{\tau_k})\, \mathbf{b}_v^{(u)}(\mathbf{x}_{\tau_k}).$$

Rearranging sums:

$$\mathrm{head}(t_q) \;=\; s_V \sum_{u=1}^{R_V} \left[\sum_{\tau_k=1}^{T} \big(\alpha_{t_q,\tau_k}\, a_v^{(u)}(\mathbf{x}_{\tau_k})\big)\, \mathbf{b}_v^{(u)}(\mathbf{x}_{\tau_k}) \right].$$

The computation of $\mathbf{b}_v^{(u)}(\mathbf{x}_{\tau_k})$ from $\mathbf{x}_{\tau_k}$ is $\mathcal{O}(T\, R_V\, d_{\text{model}} d_v)$ if contextual, or loaded if fixed (where d_v is the dimension of $\mathbf{b}_v$). Assuming $\mathbf{b}_v^{(u)}$ are available, the weighted summation for each (t_q, u) pair (inner sum over τ_k) costs $\mathcal{O}(T\, d_v)$. Summed over $t_q = 1 \ldots T$ and $u = 1 \ldots R_V$, this stage is $\mathcal{O}(T^2\, R_V\, d_v)$. The final scaling by s_V is $\mathcal{O}(T d_v)$.

C.6 Overall Complexity for Single-Head (Specialized)

The dominant costs for a single head, assuming factors $\mathbf{A}, \mathbf{B}$ are already computed from $\mathbf{X}$:

(i) **QK B-Dot Product Stage (P):** $\mathcal{O}(T^2\, R_Q\, R_K\, d_h)$ (shared calculation if multi-head).

(ii) **QK A-Factor Combination and Scaling Stage ($\mathcal{L}'$):** $\mathcal{O}(T^2\, R_Q\, R_K)$. (Scaling by $s_Q s_K$ is part of this per-element operation on scores).

(iii) **Value Aggregation Stage (including s_V scaling):** $\mathcal{O}(T^2\, R_V\, d_v)$.

Total for single head, assuming P is computed once: $\mathcal{O}\Big(T^2\, R_Q\, R_K\, d_h + T^2\, R_Q\, R_K + T^2\, R_V\, d_v\Big)$.

C.7 Multi-Head and Batch Extensions (Reuse of B-Dot Products)

QK B-Dot Products (P): Shared across heads. Cost: $\mathcal{O}\big(B\,T^2\,R_Q\,R_K\,d_h\big)$.
Per-Head QK A-Factor Combination and Scaling ($\mathcal{L}'$): Each head combines its $\mathbf{A}_Q, \mathbf{A}_K$ with P and applies $s_Q s_K$. Cost: $\mathcal{O}\big(B\,H\,T^2\,R_Q\,R_K\big)$.
Value Aggregation (including s_V): Assuming $\mathbf{B}_V$ factors are shared and $\mathbf{A}_V$ are per-head. Cost: $\mathcal{O}\big(B\,H\,T^2\,R_V\,d_h\big)$. Assuming $d_v = d_h$. Total FLOPs for specialized multi-head TPA (full sequence processing) with batch size B:

$$\mathcal{F}_{\text{TPA-specialized}} = \underbrace{\mathcal{O}\big(B\,T^2\,R_Q\,R_K\,d_h\big)}_{\text{QK } \mathbf{B}\text{-dot products } P} + \underbrace{\mathcal{O}\big(B\,H\,T^2\,R_Q\,R_K\big)}_{\text{Per-head QK } \mathbf{A}\text{-comb. + } s_Q s_K \text{ scale}} + \underbrace{\mathcal{O}\big(B\,H\,T^2\,R_V\,d_h\big)}_{\text{Value Aggregation + } s_V \text{ scale}}.$$

Discussion. By contrast, standard multi-head attention typically requires $\mathcal{F}_{\text{MHA}} = \mathcal{O}\big(2B\,T^2\,H\,d_h\big)$ FLOPs (for $\mathbf{QK}^\top$ and attention-weighted $\mathbf{V}$). The specialized TPA can be more efficient if $R_Q R_K d_h + HR_Q R_K + HR_V d_h < 2Hd_h$. Dividing by Hd_h, this condition becomes $(R_Q R_K/H) + (R_Q R_K/d_h) + R_V < 2$. For example, if $R_Q = R_K = R_V = 1$, the condition is $1/H + 1/d_h + 1 < 2$, or $1/H + 1/d_h < 1$, which is generally true for typical $H, d_h \geq 2$. Thus, with small ranks, TPA can offer computational savings. Actual speedups also depend critically on memory access patterns and kernel implementations, as demonstrated by FlashAttention-style approaches. By retaining $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ in factorized form, one can bypass the explicit materialization of these large tensors:

$$\mathbf{x}_t \to \mathbf{Q}_t,\ \mathbf{K}_\tau, \mathbf{V}_\tau \quad \text{(materialized)} \to (\mathbf{Q}\,\mathbf{K}^\top) \to \mathrm{softmax}(\mathbf{Q}\,\mathbf{K}^\top)\,\mathbf{V} \to \text{final output}.$$

Instead, the large $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors (each of size $B \times T \times H \times d_h$) are not explicitly formed. The computation is restructured into rank-based $\mathbf{B}$-dot-product computations, per-head $\mathbf{A}$-factor combinations, and appropriate scaling. The challenge lies in choosing ranks (R_Q, R_K, R_V) small enough to ensure computational benefits while maintaining model quality, and in implementing the multi-stage kernels efficiently. When ranks are sufficiently small, this specialized approach can lead to gains in both computation and memory footprint.

C.8 Inference Time Decoding of Different Attention Mechanisms

In autoregressive decoding, we generate the output for the current token $\mathbf{x}_M$ (where M is the current sequence length, previously denoted T in this section for a generic single token, but M is used in Alg. 1) given cached keys and values from $M-1$ previous tokens. We analyze the FLOPs for computing the attention output for this single query token. For all mechanisms, we consider the FLOPs for computing scaled logits and aggregating values, excluding the initial projection of the current token's hidden state $\mathbf{x}_M$ into Q, K, V representations.
MHA, MQA, and GQA. For **Multi-Head Attention (MHA)**, with H query heads and H distinct Key/Value heads:

- Logits: Each of the H query vectors (dim d_h) interacts with its corresponding K-cache (size $M \times d_h$). Cost: $H \cdot (M \cdot d_h) = MHd_h$.
- Value Aggregation: Each of the H attention patterns interacts with its V-cache (size $M \times d_h$). Cost: $H \cdot (M \cdot d_h) = MHd_h$.
- Total MHA: $2MHd_h$.

Multi-Query Attention (MQA) uses H query heads but shares a single Key/Value head ($H_{kv} = 1$).

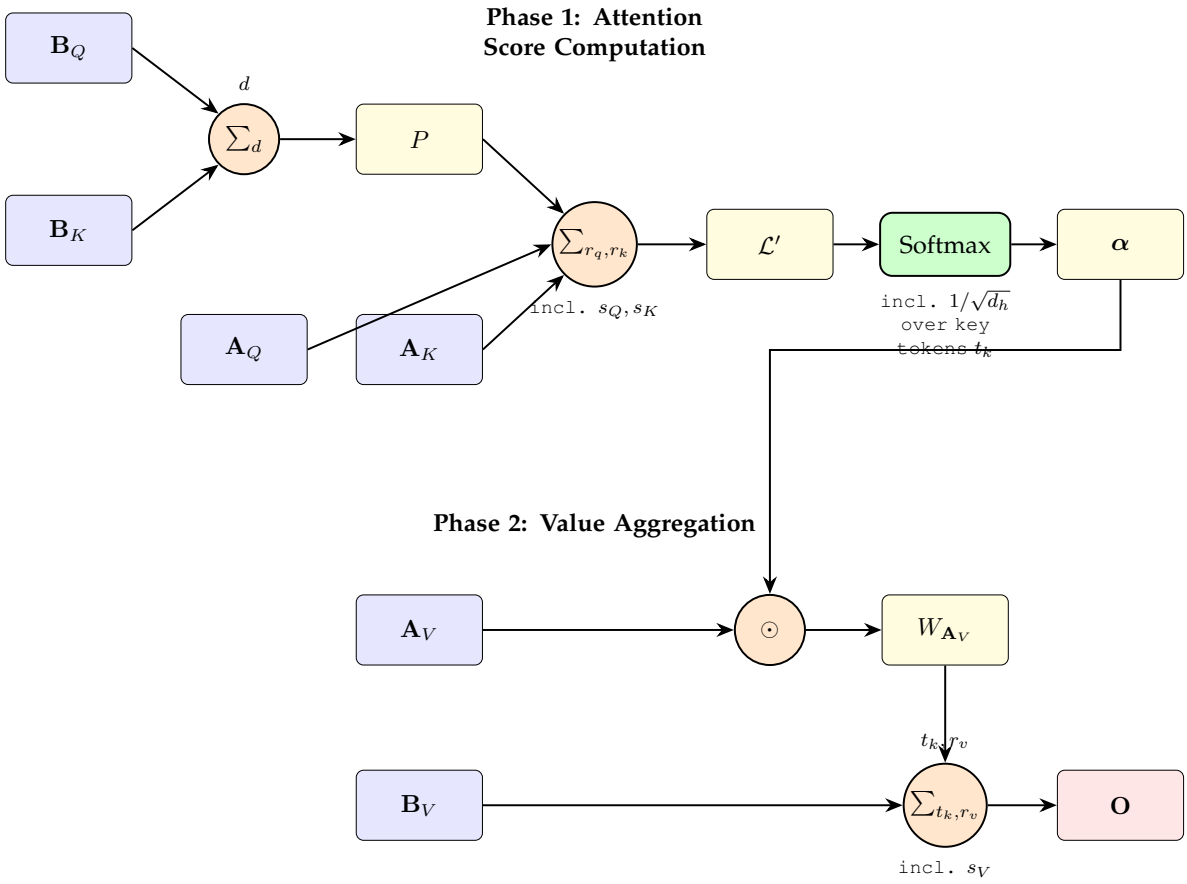


Figure 6: Data flow diagram for specialized TPA computation, avoiding materialization of full $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors. Phase 1 (Top): Attention Score Computation. Factorized $\mathbf{B}_Q, \mathbf{B}_K$ produce shared dot-products P (summing over feature dimension d). P is combined with factorized $\mathbf{A}_Q, \mathbf{A}_K$ and scaled by $s_Q s_K$ (summing over ranks r_q, r_k) to form scaled logits $\mathcal{L}'$. These are then scaled by $1/\sqrt{d_h}$ and passed through Softmax to yield attention probabilities $\boldsymbol{\alpha}$. Phase 2 (Bottom): Value Aggregation. $\boldsymbol{\alpha}$ weights the factorized $\mathbf{A}_V$ (conceptually forming $W_{\mathbf{A}_V}$ via an einsum-like combination), which then aggregates with $\mathbf{B}_V$ and is scaled by s_V (summing over key tokens t_k and value ranks r_v) to produce the final output $\mathbf{O}$. (Diagram simplifies batch dimensions B; T denotes sequence length; H is heads; R_X are ranks R_Q, R_K, R_V corresponding to indices r_q, r_k, r_v; d_h is the feature dimension d for $\mathbf{B}$ factors. Operations $\sum_X$ denote contractions over index X, and $\odot$ denotes an einsum-like combination. Scaling factors s_Q, s_K, s_V are incorporated as described.).

- Logits: Each of the H query vectors (dim d_h) interacts with the single shared K-cache (size $M \times d_h$). Cost: $H \cdot (M \cdot d_h) = MHd_h$.
- Value Aggregation: Each of the H attention patterns interacts with the single shared V-cache (size $M \times d_h$). Cost: $H \cdot (M \cdot d_h) = MHd_h$.
- Total MQA: $2MHd_h$.

Grouped-Query Attention (GQA) uses H query heads and N_g Key/Value head groups ($H_{kv} = N_g$). The H query heads are partitioned into N_g groups, each containing H/N_g query heads that share one K/V head pair within that group.

- Logits: For each of the N_g groups, H/N_g query vectors interact with the group's K-cache. Cost per group: $(H/N_g) \cdot (M \cdot d_h)$. Total for N_g groups: $N_g \cdot (H/N_g) \cdot Md_h = MHd_h$.
- Value Aggregation: Similarly, MHd_h.
- Total GQA: $2MHd_h$.

MQA and GQA significantly reduce the KV cache *size* and memory bandwidth requirements compared to MHA. While the arithmetic FLOP count for the core attention computation (dot products and weighted sums) is $2MHd_h$ for all three if they have the same number of query heads H and head dimension d_h, the practical speedups for MQA/GQA often arise from better memory access patterns due to smaller K/V caches.

MLA. Multi-Head Latent Attention (MLA), as described in Appendix H.3, uses H heads. For each head, the key $\mathbf{K}_i$ has dimension $d'_k = d_c + d_h^R$ (compressed part + RoPE part), and the value $\mathbf{V}_i^C$ has dimension $d'_v = d_c$ (only compressed part).

- Logits: H query heads (dim d'_k) interact with K-caches (dim d'_k). Cost: MHd'_k.
- Value Aggregation: H attention patterns interact with V-caches (dim d'_v). Cost: MHd'_v.
- Total MLA: $MH(d'_k + d'_v) = MH(d_c + d_h^R + d_c) = MH(2d_c + d_h^R)$.

TPA. We use the FlashTPA Decoding algorithm (Algorithm 1) for FLOPs analysis, with $N = 1$ query token, M cached items, D as feature dimension for $\mathbf{B}_Q/\boldsymbol{b}^K$ (typically d_h), and E for $\boldsymbol{b}^V$ (typically d_h). For ranks (R_Q, R_K, R_V):

- Logits computation (Steps 1-3 of Alg. 1): $\mathcal{O}(M(R_Q R_K D + HR_Q R_K + HR_K))$.
- Value aggregation (Steps 5-6 of Alg. 1): $\mathcal{O}(M(HR_V + HR_V E))$.
- Total for TPA decoding: $\mathcal{O}(M[R_K(R_Q D + HR_Q + H) + R_V H(1 + E)])$.

Example Comparison.

Let's use the ranks from FlashTPA experiments (Section 5.2): $(R_Q, R_K, R_V) = (16, 1, 1)$, and typical dimensions $D = E = d_h = 64$, $H = 32$ (e.g., for a 2048 d_{model}).

$$
\begin{aligned}
\text{MHA Logits:} \quad & M \cdot H \cdot d_h = M \cdot 32 \cdot 64 = 2048M \\
\text{MHA Value Agg.:} \quad & M \cdot H \cdot d_h = M \cdot 32 \cdot 64 = 2048M \\
\text{MHA Total:} \quad & 4096M \\
\text{TPA Logits}(R_K = 1, R_Q = 16): \quad & M(R_Q d_h + HR_Q + H) = M(16 \cdot 64 + 32 \cdot 16 + 32) \\
& = M(1024 + 512 + 32) = 1568M \\
\text{TPA Value Agg.}(R_V = 1, E = d_h): \quad & MH(1 + E) = M \cdot 32(1 + 64) = M \cdot 32 \cdot 65 = 2080M \\
\text{TPA Total:} \quad & 1568M + 2080M = 3648M
\end{aligned}
$$

For MQA and GQA, assuming $H = 32$ query heads and $d_h = 64$:

$$
\text{MQA/GQA Total:} \quad 2MHd_h = 2 \cdot M \cdot 32 \cdot 64 = 4096M
$$

For MLA, with $H = 32, d_c = 256, d_h^R = 32$:

$$\begin{aligned} \text{MLA Logits:} \quad & MH(d_c + d_h^R) = M \cdot 32 \cdot (256 + 32) = M \cdot 32 \cdot 288 = 9216M \\ \text{MLA Value Agg.:} \quad & MHd_c = M \cdot 32 \cdot 256 = 8192M \\ \text{MLA Total:} \quad & M \cdot 32 \cdot (2 \cdot 256 + 32) = M \cdot 32 \cdot 544 = 17408M \end{aligned}$$

In this configuration, TPA requires fewer FLOPs ($3648M$) than MHA ($4096M$). The FLOPs for scaled logits in TPA ($1568M$) are less than MHA's ($2048M$). The value aggregation FLOPs for TPA ($2080M$) are comparable to MHA's ($2048M$).

This reduction in FLOPs, particularly when R_K and R_V are small (e.g., 1), combined with optimized memory access patterns in algorithms like FlashTPA Decoding, contributes to TPA's competitive decoding speed, especially for long sequences. The actual end-to-end wall-clock speedup also depends on kernel fusion, hardware specifics, and the efficiency of einsum implementations, but the factorized formulation offers a clear path to reduced computational load.

D More on FlashTPA Decoding Algorithm

This section provides further details on the FlashTPA Decoding algorithm (see Algorithm 1), including the setup of factorized components.

Factorized Component Setup for FlashTPA Decoding. Let B be the batch size, N the number of query tokens (which is 1 for decoding, $N = 1$), M the current length of the KV cache, H the number of attention heads, and R_Q the rank of the query factorization. The feature dimensions for the factorized components are D (for query $\mathbf{B}_Q$ and key $\boldsymbol{b}^K$) and E (for value $\boldsymbol{b}^V$). Typically, D and E correspond to the head dimension d_h.

The query for the current token $\mathbf{x}_t$ (where t is the current time step, $N = 1$) is factorized as $\mathbf{Q}_t = \frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top \mathbf{B}_Q(\mathbf{x}_t)$, where $\mathbf{A}_Q(\mathbf{x}_t) \in \mathbb{R}^{H \times R_Q}$ (derived from the input tensor 'Aq' of shape $(B, 1, H, R_Q)$ for the current token) and $\mathbf{B}_Q(\mathbf{x}_t) \in \mathbb{R}^{R_Q \times D}$ (similarly from 'Bq' of shape $(B, 1, R_Q, D)$).

The cached keys $\mathbf{K}_s$ and values $\mathbf{V}_s$ for past tokens $s \in [1, M]$ are stored in their factorized form. For clarity in the data flow diagram (Figure 2 in the main paper) and the pseudo-code in Algorithm 1, we present the case where cached keys and values use ranks $R_K = 1$ and $R_V = 1$. The general TPA formulation allows for $R_K > 1$ and $R_V > 1$ for cached items (by storing, for example, $\mathbf{A}_K(\mathbf{x}_s) \in \mathbb{R}^{R_K \times H}$ and $\mathbf{B}_K(\mathbf{x}_s) \in \mathbb{R}^{R_K \times D}$). The einsum operations detailed in Algorithm 1 can be extended to accommodate higher ranks for keys and values by including summations over these additional rank dimensions. For $R_K = 1$ and $R_V = 1$, the cached s-th key and value are: $\mathbf{K}_s = \mathbf{a}^K(\mathbf{x}_s) \otimes \boldsymbol{b}^K(\mathbf{x}_s) \quad \in \mathbb{R}^{H \times D}$, $\mathbf{V}_s = \mathbf{a}^V(\mathbf{x}_s) \otimes \boldsymbol{b}^V(\mathbf{x}_s) \quad \in \mathbb{R}^{H \times E}$. Here, $\mathbf{a}^K(\mathbf{x}_s) \in \mathbb{R}^H$ (from the cached tensor $\mathbf{a}^K_{\text{cache}}$ of shape (B, M, H)), $\boldsymbol{b}^K(\mathbf{x}_s) \in \mathbb{R}^D$ (from $\boldsymbol{b}^K_{\text{cache}}$ of shape (B, M, D)), $\mathbf{a}^V(\mathbf{x}_s) \in \mathbb{R}^H$ (from $\mathbf{a}^V_{\text{cache}}$ of shape (B, M, H)), and $\boldsymbol{b}^V(\mathbf{x}_s) \in \mathbb{R}^E$ (from $\boldsymbol{b}^V_{\text{cache}}$ of shape (B, M, E)).

D.1 Detailed Computation Steps of FlashTPA Decoding Algorithm

The algorithm proceeds through a series of einsum operations, which correspond to specific tensor contractions:

1. **Compute $\mathbf{B}_Q$-$\boldsymbol{b}^K$ inner products ($S^{(1)}$):** For each query (batch b, query index $n = 0$), each query rank component r, and each cached key position m, we compute the dot product of the

D-dimensional feature vector of the r-th query factor $(\mathbf{B}_Q)_{b,n,r,:}$ with the D-dimensional feature vector of the m-th cached key $(\boldsymbol{b}^K_{\text{cache}})_{b,m,:}$.

$$S^{(1)}_{b,n,m,r} = \sum_{d=1}^{D} (\mathbf{B}_Q)_{b,n,r,d} \cdot (\boldsymbol{b}^K_{\text{cache}})_{b,m,d}.$$

This step captures the similarity between the feature parts of the query factors and the cached key factors.

2. **Combine with query factor $\mathbf{A}_Q$ ($S^{(2)}$):** The result $S^{(1)}$ is then combined with the head-specific query factors $\mathbf{A}_Q$. For each head h, the contributions from all query ranks r are summed up.

$$S^{(2)}_{b,n,m,h} = \sum_{r=1}^{R_Q} (\mathbf{A}_Q)_{b,n,h,r} \cdot S^{(1)}_{b,n,m,r}.$$

This yields an intermediate score for each query, cached key, and head.

3. **Incorporate cached key factor $\mathbf{a}^K_{\text{cache}}$ for full logits ($\mathcal{L}$):** The head-specific factor of the cached keys, $\mathbf{a}^K_{\text{cache}}$, is multiplied with $S^{(2)}$ to produce the final unscaled attention logits.

$$\mathcal{L}_{b,h,n,m} = S^{(2)}_{b,n,m,h} \cdot (\mathbf{a}^K_{\text{cache}})_{b,m,h}.$$

This logit $\mathcal{L}_{b,h,n,m}$ represents $s_Q^{-1} \cdot (\mathbf{Q}_t \mathbf{K}_m^\top)_h$, i.e., the dot product between the query $\mathbf{Q}_t$ and the m-th cached key $\mathbf{K}_m$ for head h, scaled by R_Q (since $s_Q = 1/R_Q$). The 'einsum' operation also rearranges dimensions to (B, H, N, M) for the subsequent softmax operation.

4. **Apply Scaling and Softmax ($\boldsymbol{\alpha}$):** The logits $\mathcal{L}$ are scaled by the product of s_Q (e.g., $1/R_Q$), s_K (e.g., $1/R_K$, which is 1 if $R_K = 1$), and s_{total} (e.g., $1/\sqrt{D}$, where D is the dimension used for dot products, typically d_h). Softmax is then applied across the M cached key positions for each head independently.

$$\alpha_{b,h,n,m} = \text{Softmax}_m \left(\mathcal{L}_{b,h,n,m} \cdot s_Q \cdot s_K \cdot s_{total} \right).$$

These are the attention probabilities.

5. **Compute weighted sum with cached value factor $\mathbf{a}^V_{\text{cache}}$ ($\mathbf{O}^{(A)}$):** The attention probabilities $\boldsymbol{\alpha}$ are used to weight the head-specific factors of the cached values, $\mathbf{a}^V_{\text{cache}}$.

$$\mathbf{O}^{(A)}_{b,n,m,h} = \alpha_{b,h,n,m} \cdot (\mathbf{a}^V_{\text{cache}})_{b,m,h}.$$

This step prepares the value components for the final aggregation. The 'einsum' operation also reorders dimensions to (B, N, M, H).

6. **Incorporate cached value factor $\boldsymbol{b}^V_{\text{cache}}$ and apply final scaling (O):** Finally, the intermediate weighted value factors $\mathbf{O}^{(A)}$ are combined with the feature-specific factors of the cached values, $\boldsymbol{b}^V_{\text{cache}}$, by summing over the M cached positions. The result is then scaled by s_V (e.g., $1/R_V$, which is 1 if $R_V = 1$).

$$\mathbf{O}_{b,n,h,e} = \left(\sum_{m=1}^{M} \mathbf{O}^{(A)}_{b,n,m,h} \cdot (\boldsymbol{b}^V_{\text{cache}})_{b,m,e} \right) \cdot s_V.$$

This produces the final output tensor of shape $(B, 1, H, E)$.

D.2 Triton FlashTPA Decoding Kernel

We implement the experiments using Triton language [56], and the detailed implementation pseudo code is displayed in Algorithm 3. The Triton FlashTPA decoding kernel supports parallelism across the number of heads, rank, and head dimensions. And in 3, we will show the experiment results with this kernel.

D.3 Additional Experimental Results

The following figures present additional speed comparisons for different embedding dimensions, with $d_h = 64$ maintained. The y-axis represents $\log_2(\text{time})$ in seconds (lower is faster), and the x-axis represents $\log_2(\text{sequence length})$.

Detailed Analysis of Figure 5 (Main Text; Embedding Dimension 2048): Figure 5 in the main paper depicts speed comparisons for an embedding dimension of 2048. The results indicate that FlashTPA (blue line) is highly competitive. Across all tested batch sizes (1 to 16) for $d_{\text{model}} = 2048$:

- MHA (orange line) is consistently the slowest mechanism, with its decoding time increasing most rapidly with sequence length.
- MQA (green line) and GQA (red line) offer significant speedups over MHA and perform very similarly to each other, often overlapping in the plots.
- MLA (purple line) demonstrates strong performance, generally being faster than MQA/GQA, particularly at longer sequence lengths.
- FlashTPA shows excellent scalability. While at very short sequence lengths (e.g., 2^{12} to 2^{13}), its performance is comparable to MQA/GQA and MLA, its decoding time increases at a notably slower rate with sequence length. Consequently, FlashTPA becomes significantly faster than MQA/GQA for sequences longer than approximately 2^{14}.
- Compared to MLA, FlashTPA is consistently among the top two performers. In many instances, particularly at sequence lengths greater than 2^{14} or 2^{15}, FlashTPA matches or slightly surpasses MLA in speed. The logarithmic scale for time suggests that these differences can be substantial in practice for very long contexts. For example, at a sequence length of 2^{19} across various batch sizes, FlashTPA often shows a visible advantage over MLA.

Figure 7 (Embedding Dimension 3072): With a larger embedding dimension of 3072, the relative performance trends observed in Figure 5 largely persist.

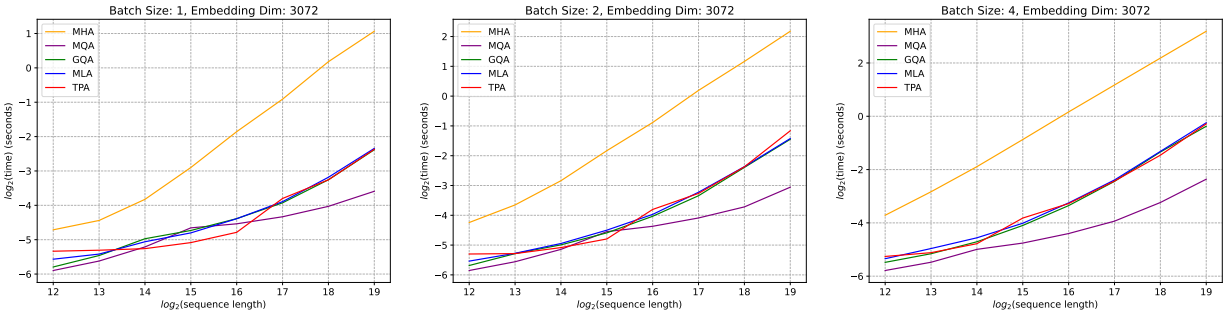
- FlashTPA (blue line) remains one of the most efficient decoding methods. MHA (orange line) is consistently the slowest, while MQA (green line) and GQA (red line) offer considerable improvements over MHA.
- MLA (purple line) and FlashTPA are the top two performers. FlashTPA consistently matches or exceeds the speed of MLA, particularly at longer sequence lengths (e.g., beyond 2^{15} or 2^{16} depending on the batch size). Its advantage often becomes more pronounced at the longest sequences tested (2^{19}). For instance, in batch size 1, TPA is clearly faster than MLA for sequence lengths 2^{16} and above. A similar trend is seen across other batch sizes, where TPA maintains a competitive edge or becomes superior at longer contexts.

This suggests that FlashTPA's efficiency is well-maintained even as the model's embedding dimension increases.

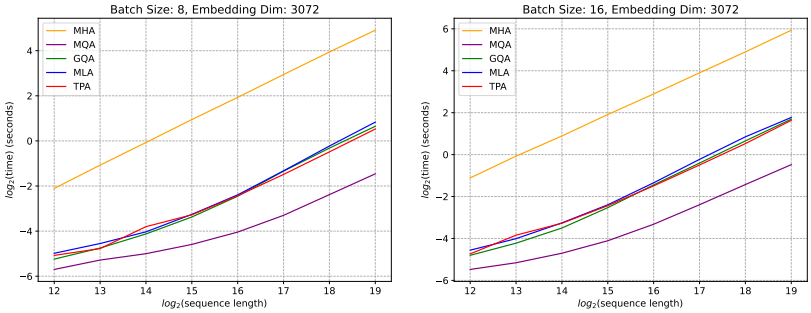
Figure 8 (Embedding Dimension 1024): For a smaller embedding dimension of 1024, similar trends are observed:

- FlashTPA (blue line) is highly competitive. MHA (orange line) remains the least performant. MQA (green line) and GQA (red line) are faster than MHA.
- At very short sequence lengths (around 2^{12} to 2^{13}), MQA/GQA can be slightly faster than or comparable to TPA and MLA, especially for smaller batch sizes (e.g., Batch Size 1).
- However, as sequence length increases, both MLA (purple line) and FlashTPA demonstrate superior scalability. FlashTPA generally matches or outperforms MLA, particularly for sequences longer than 2^{15}. For example, with a batch size of 16, TPA shows a clear speed advantage over MLA for sequence lengths 2^{16} and greater.

These results across different embedding dimensions highlight the robustness of FlashTPA's decoding speed advantages, especially for long sequences where it consistently ranks as one of the fastest, if not the fastest, attention mechanism among those tested.

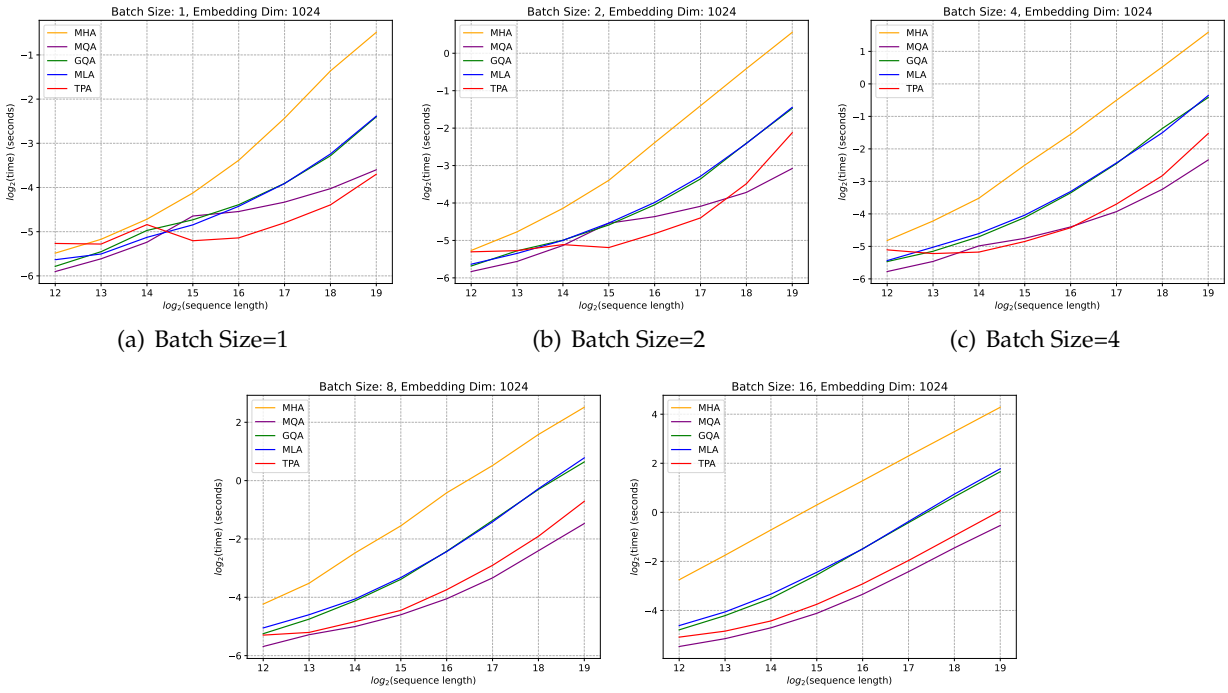


(a) Batch Size=1 (b) Batch Size=2 (c) Batch Size=4



(d) Batch Size=8 (e) Batch Size=16

Figure 7: Decoding time comparison of different attention mechanisms with an embedding dimension of 3072 and $d_h = 64$.



(a) Batch Size=1 (b) Batch Size=2 (c) Batch Size=4

(d) Batch Size=8 (e) Batch Size=16

Figure 8: Decoding time comparison of different attention mechanisms with an embedding dimension of 1024 and $d_h = 64$.

Algorithm 3 Triton FlashTPA Decoding Kernel

Require: Input Tensors: $\mathbf{A}_Q(B, N, H, R_Q)$, $\mathbf{a}^K(B, M, H)$, $\mathbf{a}^V(B, M, H)$, $\mathbf{B}_Q(B, N, R_Q, D)$, $\boldsymbol{b}^K(B, M, D)$, $\boldsymbol{b}^V(B, M, E)$
Require: Scaling factors: $s_{\text{total}}, s_Q, s_K, s_V$; Dimensions: $B, N(=1), M, H, R_Q, D, E$
Require: Kernel Block dims: B_H, B_R, B_D, B_E; Sequence Blocking: $M_{\text{block}}, M_{\text{chunk}}$
Require: Program IDs: $p_{id_B}, p_{id_H}, p_{id_M}$
Ensure: Partial Output $\mathbf{O}_{\text{partial}}(B, \text{Num}_M, N, H, E)$, Log-Sum-Exp $\mathbf{LSE}_{\text{partial}}(B, \text{Num}_M, H)$

$b \leftarrow p_{id_B}$; $h_{\text{start}} \leftarrow p_{id_H} \cdot B_H$
$m_{\text{block\_start}} \leftarrow p_{id_M} \cdot M_{\text{block}}$; $m_{\text{block\_end}} \leftarrow \min((p_{id_M} + 1) \cdot M_{\text{block}}, M)$
▷ B_H, B_R, B_D, B_E are tile sizes for dimensions H, R, D, E respectively.

▷ Initialize accumulators for the head block
$\mathbf{o}_{\text{accum}} \leftarrow \mathbf{0}^{(E \times B_H)}$; $\mathbf{m}_{\text{max}} \leftarrow -\infty^{(B_H)}$; $\mathbf{s}_{\text{exp\_sum}} \leftarrow \mathbf{0}^{(B_H)}$; $c_{\text{scale}} \leftarrow s_{\text{total}} \cdot s_Q \cdot s_K$

▷ Load query factors (fixed for this program as N=1)
Load $\mathbf{A}_{Q,\text{local}}^{(R_Q \times B_H)}$ from $\mathbf{A}_Q[b, 0, h_{\text{start}} \ldots, :]$
Load $\mathbf{B}_{Q,\text{local}}^{(D \times R_Q)}$ from $\mathbf{B}_Q[b, 0, :, :]$ ▷ Dimensions may be transposed after loading for matmul

▷ Iterate over M_{chunk}-sized chunks within the K/V block
for $m_{\text{chunk\_start}}$ from $m_{\text{block\_start}}$ to $m_{\text{block\_end}} - 1$ step M_{chunk} **do**
    $m_{\text{chunk\_end}} \leftarrow \min(m_{\text{chunk\_start}} + M_{\text{chunk}}, m_{\text{block\_end}})$
    $M_{\text{curr\_chunk}} \leftarrow m_{\text{chunk\_end}} - m_{\text{chunk\_start}}$
    ▷ Load K/V factors for the current chunk
    Load $\mathbf{a}_{\text{chunk}}^K(M_{\text{curr\_chunk}}, B_H)$; $\mathbf{a}_{\text{chunk}}^V(M_{\text{curr\_chunk}}, B_H)$; $\boldsymbol{b}_{\text{chunk}}^K(M_{\text{curr\_chunk}}, D)$; $\boldsymbol{b}_{\text{chunk}}^V(E, M_{\text{curr\_chunk}})$ ▷ Layouts optimized for memory access and matmuls
    $\boldsymbol{b}_{\text{chunk}}^V \leftarrow \boldsymbol{b}_{\text{chunk}}^V \cdot s_V$
    ▷ Core TPA Score Calculation for the chunk
    $S1_{\text{chunk}} \leftarrow \text{MatMul}(\boldsymbol{b}_{\text{chunk}}^K, \mathbf{B}_{Q,\text{local}})$ ▷ Shape: $(M_{\text{curr\_chunk}}, R_Q)$
    $S2_{\text{chunk}} \leftarrow \text{MatMul}(S1_{\text{chunk}}, \mathbf{A}_{Q,\text{local}})$ ▷ Shape: $(M_{\text{curr\_chunk}}, B_H)$
    $S3_{\text{chunk}} \leftarrow S2_{\text{chunk}} \odot \mathbf{a}_{\text{chunk}}^K \cdot c_{\text{scale}}$ ▷ Shape: $(M_{\text{curr\_chunk}}, B_H)$
    ▷ Online Softmax Update for the chunk
    $\mathbf{m}_{\text{max\_local}} \leftarrow \max_{axis=0}(S3_{\text{chunk}})$ ▷ Shape: (B_H)
    $\mathbf{m}_{\text{max\_new}} \leftarrow \max(\mathbf{m}_{\text{max}}, \mathbf{m}_{\text{max\_local}})$
    $\mathbf{p}_{\text{num}} \leftarrow \exp(S3_{\text{chunk}} - \mathbf{m}_{\text{max\_new}}[\text{None}, :])$
    $\mathbf{s}_{\text{exp\_sum\_local}} \leftarrow \sum_{axis=0}(\mathbf{p}_{\text{num}})$
    $\mathbf{p}_{\text{weighted\_av}} \leftarrow (\mathbf{p}_{\text{num}} / \mathbf{s}_{\text{exp\_sum\_local}}[\text{None}, :]) \odot \mathbf{a}_{\text{chunk}}^V$
    $\mathbf{o}_{\text{chunk}} \leftarrow \text{MatMul}(\boldsymbol{b}_{\text{chunk}}^V, \mathbf{p}_{\text{weighted\_av}})$ ▷ Shape: (E, B_H)
    ▷ Update global (M-block level) accumulators
    $\mathbf{s}_{\text{exp\_sum\_prev\_rescaled}} \leftarrow \mathbf{s}_{\text{exp\_sum}} \cdot \exp(\mathbf{m}_{\text{max}} - \mathbf{m}_{\text{max\_new}})$
    $\mathbf{s}_{\text{exp\_sum}} \leftarrow \mathbf{s}_{\text{exp\_sum\_prev\_rescaled}} + \mathbf{s}_{\text{exp\_sum\_local}}$
    $\text{ratio} \leftarrow \mathbf{s}_{\text{exp\_sum\_local}} / \mathbf{s}_{\text{exp\_sum}}$ ▷ This is $\mathbf{s}_{\text{exp\_sum\_local}} / \mathbf{s}_{\text{exp\_sum\_new}}$
    $\mathbf{o}_{\text{accum}} \leftarrow (1 - \text{ratio}) \cdot \mathbf{o}_{\text{accum}} + \text{ratio} \cdot \mathbf{o}_{\text{chunk}}$
    $\mathbf{m}_{\text{max}} \leftarrow \mathbf{m}_{\text{max\_new}}$
end for

▷ Store partial results for this program's (batch, head_block, M_block)
Store $\mathbf{o}_{\text{accum}}$ into $\mathbf{O}_{\text{partial}}[b, p_{id_M}, 0, h_{\text{start}} \ldots, :]$
$\mathbf{LSE}_{\text{val}} \leftarrow \log(\mathbf{s}_{\text{exp\_sum}}) + \mathbf{m}_{\text{max}}$
Store $\mathbf{LSE}_{\text{val}}$ into $\mathbf{LSE}_{\text{partial}}[b, p_{id_M}, h_{\text{start}} \ldots]$

E Higher-Order Tensor Product Attention

All prior discussions have focused on TPA where the query, key, and value matrices (e.g., $\mathbf{Q}_t \in \mathbb{R}^{h \times d_h}$) are formed as a sum of R_Q components. Each component is an outer product of two context-dependent vectors, one spanning the head dimension ($\mathbb{R}^h$) and the other spanning the feature-per-head dimension ($\mathbb{R}^{d_h}$), as detailed in Section 3.1 (e.g., $\mathbf{Q}_t = \frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top \mathbf{B}_Q(\mathbf{x}_t)$ implies $\mathbf{Q}_t = \sum_r \mathbf{a}_r \boldsymbol{b}_r^\top$ where $\mathbf{a}_r$ are columns of $\mathbf{A}_Q^\top$ and $\boldsymbol{b}_r^\top$ are rows of $\mathbf{B}_Q$). We now generalize this by introducing additional latent factors in the construction of the feature-per-head vectors, leading to what we term *higher-order* TPA. This approach allows for more complex interactions in forming these feature vectors.

For instance, in a third-order factorization, the query tensor $\mathbf{Q}_t \in \mathbb{R}^{h \times d_h}$ for a single token t is constructed as:

$$\mathbf{Q}_t = \frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q(\mathbf{x}_t) \otimes \mathrm{vec}\big(\mathbf{b}_r^Q(\mathbf{x}_t) \otimes \mathbf{c}_r^Q(\mathbf{x}_t)\big),$$

where $\mathbf{a}_r^Q(\mathbf{x}_t) \in \mathbb{R}^h$. The term $\mathbf{b}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_b}$ and the newly introduced factor $\mathbf{c}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_c}$ first form a matrix $\mathbf{b}_r^Q(\mathbf{x}_t) \otimes \mathbf{c}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_b \times d_c}$ via an outer product (as defined in Section 2). This matrix is then vectorized by $\mathrm{vec}(\cdot)$ into a column vector of dimension $d_h = d_b d_c$. The final query $\mathbf{Q}_t$ is formed by the sum of outer products between $\mathbf{a}_r^Q(\mathbf{x}_t)$ and these resulting d_h-dimensional vectors. Analogous expansions apply to $\mathbf{K}_t$ and $\mathbf{V}_t$.

The additional factor $\mathbf{c}_r^Q(\mathbf{x}_t)$ can be viewed as a learnable, context-dependent modulation or gating term for the features generated by $\mathbf{b}_r^Q(\mathbf{x}_t)$.

$$\mathbf{b}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_b}, \quad \mathbf{c}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_c}, \quad d_h = d_b d_c.$$

This higher-order construction can enhance expressiveness. While introducing $\mathbf{c}_r^Q$ increases the parameter count for the factors, it might allow for the use of smaller base ranks (R_Q, R_K, R_V) to achieve comparable representational power, thus offering a different design choice. One could also explore tying or sharing $\mathbf{c}_r^Q$ across queries, keys, and values to manage parameter overhead.

From a memory perspective during inference, higher-order TPA maintains the benefit of factorized KV caching. Only the constituent factors $\mathbf{a}_K(\mathbf{x}_t), \mathbf{b}_K(\mathbf{x}_t), \mathbf{c}_K(\mathbf{x}_t)$ (and similarly for values) for each past token need to be stored. A trade-off arises between model capacity and the overhead of memory and computation. Higher-order tensor decompositions can provide additional flexibility and potentially increased capacity.

E.1 RoPE Compatibility in Higher-Order TPA

Rotary positional embeddings (RoPE) remain compatible with higher-order factorizations. In second-order TPA, RoPE applies rotations to the d_h-dimensional feature vectors. This compatibility extends to higher-order TPA. Consider the case where RoPE is intended to primarily rotate feature pairs derived from the $\mathbf{b}_r^Q(\mathbf{x}_t)$ components, while the structural influence of $\mathbf{c}_r^Q(\mathbf{x}_t)$ components on the d_h-dimensional vector is preserved. More formally, RoPE acts on the d_h-dimensional vector $\mathrm{vec}(\mathbf{b}_r^Q \otimes \mathbf{c}_r^Q)$ such that the transformation is equivalent to rotating $\mathbf{b}_r^Q$ to $\widetilde{\mathbf{b}}_r^Q = \mathbf{R}_t \mathbf{b}_r^Q$ (where $\mathbf{R}_t$ is the RoPE rotation matrix for d_b dimensions) and then forming $\mathrm{vec}(\widetilde{\mathbf{b}}_r^Q \otimes \mathbf{c}_r^Q)$. This is achieved by a

specific RoPE transformation matrix $\mathbf{T}_t$ acting on the full d_h-dimensional vector, as stated in the following theorem.

Theorem E.1 (RoPE Compatibility in Higher-Order TPA)**.** Consider the higher-order (3-order) Tensor Product Attention (TPA) query factorization

$$\mathbf{Q}_t = \frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q(\mathbf{x}_t) \otimes \mathrm{vec}\big(\mathbf{b}_r^Q(\mathbf{x}_t) \otimes \mathbf{c}_r^Q(\mathbf{x}_t)\big) \in \mathbb{R}^{h \times d_h},$$

where $\mathbf{a}_r^Q(\mathbf{x}_t) \in \mathbb{R}^h$, $\mathbf{b}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_b}$, $\mathbf{c}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_c}$, with $d_h = d_b d_c$. Define the RoPE-transformed query as $\widetilde{\mathbf{Q}}_t = \mathrm{RoPE}_t\big(\mathbf{Q}_t\big) = \mathbf{Q}_t \mathbf{T}_t$, where

$$\mathbf{T}_t = \mathbf{I}_{d_c} \otimes (\mathbf{R}_t)^\top = \begin{pmatrix} (\mathbf{R}_t)^\top & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\mathbf{R}_t)^\top & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & (\mathbf{R}_t)^\top \end{pmatrix} \in \mathbb{R}^{d_h \times d_h},$$

$\mathbf{I}_{d_c}$ is the identity matrix of size $d_c \times d_c$, and $\mathbf{R}_t \in \mathbb{R}^{d_b \times d_b}$ ($d_b \in \mathbb{Z}_+$ is even) is the standard RoPE block-diagonal matrix composed of 2×2 rotation matrices:

$$\mathbf{R}_t = \begin{pmatrix} \cos(t\theta_1) & -\sin(t\theta_1) & & & & & & \\ \sin(t\theta_1) & \cos(t\theta_1) & & & & & & \\ & & \cos(t\theta_2) & -\sin(t\theta_2) & & & & \\ & & \sin(t\theta_2) & \cos(t\theta_2) & & & & \\ & & & & \ddots & & & \\ & & & & & & \cos(t\theta_{d_b/2}) & -\sin(t\theta_{d_b/2}) \\ & & & & & & \sin(t\theta_{d_b/2}) & \cos(t\theta_{d_b/2}) \end{pmatrix},$$

for $t \in \{1, \ldots, T\}$ and $j \in \{1, \ldots, d_b/2\}$. The transformation $\mathbf{T}_t = \mathbf{I}_{d_c} \otimes (\mathbf{R}_t)^\top$ operates on the d_h-dimensional vectorized features by post-multiplication. This structure of $\mathbf{T}_t$ ensures that the rotation effectively applied to the $\mathbf{b}_r^Q(\mathbf{x}_t)$ component (which is a column vector) corresponds to a pre-multiplication by $\mathbf{R}_t$, as detailed in the proof (Appendix F.2). This preserves the structure induced by $\mathbf{c}_r^Q(\mathbf{x}_t)$ while rotating $\mathbf{b}_r^Q(\mathbf{x}_t)$.

Under these conditions, the RoPE-transformed query $\mathrm{RoPE}_t\big(\mathbf{Q}_t\big)$ admits a higher-order TPA factorization of the same rank R_Q:

$$\frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q(\mathbf{x}_t) \otimes \mathrm{vec}\Big(\widetilde{\mathbf{b}}_r^Q(\mathbf{x}_t) \otimes \mathbf{c}_r^Q(\mathbf{x}_t)\Big) = \mathrm{RoPE}_t\big(\mathbf{Q}_t\big), \tag{E.1}$$

where $\widetilde{\mathbf{b}}_r^Q(\mathbf{x}_t) = \mathbf{R}_t \mathbf{b}_r^Q(\mathbf{x}_t)$.

Please see Appendix F.2 for the proof. For fourth-order or higher, this result still holds.

To assess its empirical performance, we implemented third-order TPA. Table 4 lists the evaluation results for a small model. These results provide an initial indication of its viability. A comprehensive comparison with second-order TPA variants of similar parameter counts or ranks would be necessary to fully evaluate the trade-offs.

Table 4: The evaluation results of small models with third-order TPA pre-trained using FineWeb-Edu 100B dataset with lm-evaluation-harness. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Few-shot	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
0-shot	49.24	24.91	57.06	34.01	31.80	63.33	50.59	23.23	66.9	44.56
2-shot	53.37	25.34	48.78	34.00	29.20	62.79	52.33	26.41	75.3	45.28

F Proofs of Theorems

F.1 Proof of Theorem 3.1

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 3.1. □

F.2 Proof of Theorem E.1

Theorem E.1 addresses the compatibility of RoPE with higher-order (specifically, 3rd-order) Tensor Product Attention. The theorem considers the query factorization:

$$\mathbf{Q}_t = \frac{1}{R_Q}\sum_{r=1}^{R_Q}\mathbf{a}_r^Q(\mathbf{x}_t)\otimes\mathrm{vec}\big(\mathbf{b}_r^Q(\mathbf{x}_t)\otimes\mathbf{c}_r^Q(\mathbf{x}_t)\big) \in \mathbb{R}^{h\times d_h},$$

where $\mathbf{a}_r^Q(\mathbf{x}_t) \in \mathbb{R}^h$ (column vector), $\mathbf{b}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_b}$ (column vector), $\mathbf{c}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_c}$ (column vector), and $d_h = d_b d_c$. The term $\mathbf{b}_r^Q(\mathbf{x}_t)\otimes\mathbf{c}_r^Q(\mathbf{x}_t)$ is interpreted as the matrix $\mathbf{M}_r = \mathbf{b}_r^Q(\mathbf{x}_t)(\mathbf{c}_r^Q(\mathbf{x}_t))^\top \in \mathbb{R}^{d_b\times d_c}$. The notation $\mathbf{a}\otimes\mathbf{v}$ for $\mathbf{a}\in\mathbb{R}^h$ and $\mathbf{v}\in\mathbb{R}^{d_h}$ (column vectors) implies the outer product $\mathbf{a}\mathbf{v}^\top$. Thus, $\mathbf{Q}_t = \frac{1}{R_Q}\sum_{r=1}^{R_Q}\mathbf{a}_r^Q(\mathbf{x}_t)(\mathrm{vec}(\mathbf{M}_r))^\top$.

The RoPE-transformed query is defined as $\widetilde{\mathbf{Q}}_t = \mathrm{RoPE}_t\big(\mathbf{Q}_t\big) = \mathbf{Q}_t\mathbf{T}_t$. Crucially, for the theorem's conclusion to hold as intended (i.e., that the $\mathbf{b}_r^Q$ component is transformed by pre-multiplication with the standard RoPE matrix $\mathbf{R}_t$), the global transformation matrix $\mathbf{T}_t \in \mathbb{R}^{d_h\times d_h}$ (that post-multiplies $\mathbf{Q}_t$) is given by:

$$\mathbf{T}_t = \mathbf{I}_{d_c}\otimes(\mathbf{R}_t)^\top,$$

where $\mathbf{I}_{d_c}$ is the $d_c \times d_c$ identity matrix, and $\mathbf{R}_t \in \mathbb{R}^{d_b \times d_b}$ is the standard RoPE block-diagonal matrix that pre-multiplies d_b-dimensional column vectors (as defined explicitly in the theorem statement in Section E).

The theorem claims that, under these conditions, $\widetilde{\mathbf{Q}}_t$ admits a higher-order TPA factorization:

$$\widetilde{\mathbf{Q}}_t = \frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q(\mathbf{x}_t) \otimes \mathrm{vec}\Big(\widetilde{\mathbf{b}}_r^Q(\mathbf{x}_t) \otimes \mathbf{c}_r^Q(\mathbf{x}_t)\Big),$$

where $\widetilde{\mathbf{b}}_r^Q(\mathbf{x}_t) = \mathbf{R}_t \mathbf{b}_r^Q(\mathbf{x}_t)$.

Proof. Let $\mathbf{a}_r^Q \equiv \mathbf{a}_r^Q(\mathbf{x}_t)$, $\mathbf{b}_r^Q \equiv \mathbf{b}_r^Q(\mathbf{x}_t)$, and $\mathbf{c}_r^Q \equiv \mathbf{c}_r^Q(\mathbf{x}_t)$ for brevity. Let $\mathbf{M}_r = \mathbf{b}_r^Q (\mathbf{c}_r^Q)^\top \in \mathbb{R}^{d_b \times d_c}$. Let $\mathbf{v}_r = \mathrm{vec}(\mathbf{M}_r) \in \mathbb{R}^{d_h}$ be the column vector obtained by stacking the columns of $\mathbf{M}_r$. The query tensor is $\mathbf{Q}_t = \frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q (\mathbf{v}_r)^\top$.

The RoPE transformation is $\widetilde{\mathbf{Q}}_t = \mathbf{Q}_t \mathbf{T}_t$. Substituting the factorization and the revised definition of $\mathbf{T}_t$:

$$\begin{aligned}\widetilde{\mathbf{Q}}_t &= \left(\frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q (\mathbf{v}_r)^\top \right) (\mathbf{I}_{d_c} \otimes (\mathbf{R}_t)^\top) \\ &= \frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q \left((\mathbf{v}_r)^\top (\mathbf{I}_{d_c} \otimes (\mathbf{R}_t)^\top) \right).\end{aligned}$$

Let's analyze the transformed vector part for the r-th component: $(\mathbf{v}_r)^\top (\mathbf{I}_{d_c} \otimes (\mathbf{R}_t)^\top)$. This row vector is the transpose of $((\mathbf{I}_{d_c} \otimes (\mathbf{R}_t)^\top)^\top \mathbf{v}_r)$. Let's compute the pre-multiplying matrix:

$$((\mathbf{I}_{d_c} \otimes (\mathbf{R}_t)^\top)^\top = (\mathbf{I}_{d_c})^\top \otimes ((\mathbf{R}_t)^\top)^\top = \mathbf{I}_{d_c} \otimes \mathbf{R}_t.$$

So, the column vector transformation is $(\mathbf{I}_{d_c} \otimes \mathbf{R}_t)\mathbf{v}_r$. Substitute $\mathbf{v}_r = \mathrm{vec}(\mathbf{M}_r) = \mathrm{vec}(\mathbf{b}_r^Q (\mathbf{c}_r^Q)^\top)$:

$$(\mathbf{I}_{d_c} \otimes \mathbf{R}_t)\, \mathrm{vec}(\mathbf{b}_r^Q (\mathbf{c}_r^Q)^\top).$$

We use the Kronecker product identity: $(\mathbf{B_0}^\top \otimes \mathbf{A_0})\, \mathrm{vec}(\mathbf{X_0}) = \mathrm{vec}(\mathbf{A_0 X_0 B_0})$. To match our expression $(\mathbf{I}_{d_c} \otimes \mathbf{R}_t)\, \mathrm{vec}(\mathbf{M}_r)$, we identify: $\mathbf{A_0} = \mathbf{R}_t$, $\mathbf{B_0}^\top = \mathbf{I}_{d_c} \implies \mathbf{B_0} = \mathbf{I}_{d_c}$, $\mathbf{X_0} = \mathbf{M}_r = \mathbf{b}_r^Q (\mathbf{c}_r^Q)^\top$. Applying the identity, we get:

$$\mathrm{vec}\left(\mathbf{R}_t (\mathbf{b}_r^Q (\mathbf{c}_r^Q)^\top) \mathbf{I}_{d_c} \right) = \mathrm{vec}\left((\mathbf{R}_t \mathbf{b}_r^Q)(\mathbf{c}_r^Q)^\top \right).$$

Let $\widetilde{\mathbf{b}}_r^Q = \mathbf{R}_t \mathbf{b}_r^Q$. This is precisely the transformation for the $\mathbf{b}_r^Q$ component as claimed in the theorem. So the transformed column vector is $\mathrm{vec}(\widetilde{\mathbf{b}}_r^Q (\mathbf{c}_r^Q)^\top)$. The corresponding row vector in the sum for $\widetilde{\mathbf{Q}}_t$ is therefore $(\mathrm{vec}(\widetilde{\mathbf{b}}_r^Q (\mathbf{c}_r^Q)^\top))^\top$.

Substituting this back into the expression for $\widetilde{\mathbf{Q}}_t$:

$$\widetilde{\mathbf{Q}}_t = \frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q (\mathrm{vec}(\widetilde{\mathbf{b}}_r^Q (\mathbf{c}_r^Q)^\top))^\top.$$

This is equivalent to the theorem's claimed factorization, using the definition $\mathbf{a} \otimes \mathbf{col\_vec} = \mathbf{a}(\mathbf{col\_vec})^\top$:

$$\widetilde{\mathbf{Q}}_t = \frac{1}{R_Q} \sum_{r=1}^{R_Q} \mathbf{a}_r^Q \otimes \mathrm{vec}\big(\widetilde{\mathbf{b}}_r^Q \otimes \mathbf{c}_r^Q\big),$$

where $\widetilde{\mathbf{b}}_r^Q = \mathbf{R}_t \mathbf{b}_r^Q$. This completes the proof, showing that RoPE can be consistently applied to higher-order TPA representations if the global RoPE transformation matrix $\mathbf{T}_t$ (that post-multiplies $\mathbf{Q}_t$) is appropriately defined as $\mathbf{I}_{d_c} \otimes (\mathbf{R}_t)^\top$, ensuring that the standard RoPE matrix $\mathbf{R}_t$ effectively pre-multiplies the $\mathbf{b}_r^Q$ component. □

G More Related Work

Transformers and Attention. As a sequence-to-sequence architecture, Transformer [59] 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 [9, 48, 15, 29, 26, 30], kernel-based projections [10], and linearized transformers [58, 24, 43, 68, 53, 66]. To decrease memory usage and circumvent the limitation of memory bandwidth in training, [45] proposed Multi-Query Attention (MQA) where multiple query heads share the same key head and value head. To tackle the issue of quality degradation and instability in training, Grouped-Query Attention (GQA) [2] divides queries into several groups, and each group of queries shares a single key head and value head. Recently, DeepSeek-V2 [31] 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. Concurrently, [20] proposed Multi-matrix Factorization Attention (MFA), which can be simply seen as MQA with low-rank factorized Q. Compared to the approaches above, TPA applied contextual tensor decompositions to represent queries, keys, and values activations compactly, achieving better reduction on the size of KV cache with improved performance.

KV Cache Optimization. During the auto-regressive inference of Transformers, key and value (KV) tensors from previous tokens are cached to avoid recomputation, a technique first proposed by [39]. This Key-Value (KV) cache, while crucial for efficiency, consumes significant memory and can introduce latency bottlenecks due to memory bandwidth limitations [1]. Consequently, various studies have explored methods to mitigate these issues. These include KV cache eviction strategies that discard less significant tokens [69, 61, 7, 1], dynamic sparse attention mechanisms focusing on selected keys and values [41, 54, 49], offloading the KV cache to CPU memory [16, 25, 52], and quantizing the KV cache [60, 33, 18]. In contrast to these approaches, TPA focuses on reducing the intrinsic size of the KV cache by employing tensor-decomposed key and value representations.

Low-Rank Factorizations. Low-rank approximations are widely used to compress model parameters and reduce computational complexity. Notable examples include LoRA [19], which factorizes weight updates during fine-tuning, and its derivatives tailored for various training scenarios such as efficient pretraining (ReLoRA [27], MoRA [21]), long-context training (LongLoRA [8], SinkLoRA [65]), and continual training (InfLoRA [28], GS-LoRA [70], I-LoRA [40]). These methods generally produce static low-rank expansions that are independent of the input context. Theoretical justifications for the expressiveness of low-rank approximations have been provided by [37, 64]. Initialization strategies for these factorization matrices have also been explored: OLoRA [6] utilizes QR-decomposition of pretrained weights for improved language model performance, while

LoLDU [47] employs LDU-decomposition to accelerate LoRA training. Furthermore, AdaLoRA [67] uses Singular Value Decomposition (SVD) on pretrained weights and introduces parameter importance scores to dynamically adjust ranks. TPA, in contrast, constructs Q, K, and V tensors using contextually-aware factorizations, allowing for dynamic adaptation based on the input.

H More on Attention Mechanisms

H.1 Multi-Query Attention (MQA)

Multi-Query Attention (MQA) [45] significantly reduces memory usage, particularly for the KV cache, by sharing a single key and value projection across all attention heads, while each head maintains a unique query projection. Given a sequence of input embeddings $\mathbf{X} \in \mathbb{R}^{T \times d_{\text{model}}}$, the query, shared key, and shared value tensors are computed as:

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

Thus, each head i uses a distinct query projection $\mathbf{Q}_i \in \mathbb{R}^{T \times d_h}$ but shares the common key $\mathbf{K}_{\text{shared}} \in \mathbb{R}^{T \times d_h}$ and value $\mathbf{V}_{\text{shared}} \in \mathbb{R}^{T \times d_h}$ tensors. The weight matrices are:

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

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 key and value projections, MQA substantially reduces memory demands, especially for the KV cache during autoregressive inference. However, this comes at the cost of reduced model expressivity, as all heads must utilize the same key and value representations.

H.2 Grouped Query Attention (GQA)

Grouped Query Attention (GQA) [2] generalizes Multi-Head Attention (MHA) and MQA by dividing the total h attention heads into G groups. Within each group, heads share a common key and value projection, while each head maintains its own unique query projection. Formally, let $g(i)$ denote the group index for head $i \in \{1, \ldots, h\}$, where $g(i) \in \{1, \ldots, G\}$. The projections are:

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

and

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

Here, $\boldsymbol{W}_g^K$ and $\boldsymbol{W}_g^V$ are the shared weight matrices for group g, each in $\mathbb{R}^{d_{\text{model}} \times d_h}$, and $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_h}$ is the query weight matrix for 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 varying G from 1 (equivalent to MQA) to h (equivalent to MHA), GQA offers a trade-off between memory efficiency and model capacity.

H.3 Multi-head Latent Attention (MLA)

Multi-head Latent Attention (MLA), as used in DeepSeek-V2 [31] and DeepSeek-V3 [32], introduces low-rank compression for keys and values to reduce KV caching costs during inference.

$$\begin{aligned}
\mathbf{C}^{KV} &= \mathbf{X}\boldsymbol{W}^{DKV}, \\
\text{Concat}\big(\mathbf{K}_1^C, \mathbf{K}_2^C, \ldots, \mathbf{K}_h^C\big) &= \mathbf{K}^C = \mathbf{C}^{KV}\boldsymbol{W}^{UK}, \\
\mathbf{K}^R &= \text{RoPE}\big(\mathbf{X}\boldsymbol{W}^{KR}\big), \\
\mathbf{K}_i &= \text{Concat}\big(\mathbf{K}_i^C, \mathbf{K}^R\big), \\
\text{Concat}\big(\mathbf{V}_1^C, \mathbf{V}_2^C, \ldots, \mathbf{V}_h^C\big) &= \mathbf{V}^C = \mathbf{C}^{KV}\boldsymbol{W}^{UV},
\end{aligned}$$

Here, $\boldsymbol{W}^{DKV} \in \mathbb{R}^{d_{\text{model}} \times d_c}$ projects to a compressed dimension d_c, $\boldsymbol{W}^{UK} \in \mathbb{R}^{d_c \times (d_h h)}$ up-projects the compressed keys, $\boldsymbol{W}^{KR} \in \mathbb{R}^{d_{\text{model}} \times d_h^R}$ projects to a residual key component for RoPE, and $\boldsymbol{W}^{UV} \in \mathbb{R}^{d_c \times (d_h h)}$ up-projects the compressed values. $\mathbf{C}^{KV} \in \mathbb{R}^{T \times d_c}$ is the shared compressed KV latent (where $d_c \ll d_h h$). The RoPE transformation is applied to a separate key embedding $\mathbf{K}^R \in \mathbb{R}^{T \times d_h^R}$. Thus, only $\mathbf{C}^{KV}$ and $\mathbf{K}^R$ are cached, reducing KV memory usage while largely preserving performance compared to standard MHA [59].

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

$$\begin{aligned}
\mathbf{C}^Q &= \mathbf{X}\boldsymbol{W}^{DQ}, \\
\text{Concat}\big(\mathbf{Q}_1^C, \mathbf{Q}_2^C, \ldots, \mathbf{Q}_h^C\big) &= \mathbf{Q}^C = \mathbf{C}^Q\boldsymbol{W}^{UQ}, \\
\text{Concat}\big(\mathbf{Q}_1^R, \mathbf{Q}_2^R, \ldots, \mathbf{Q}_h^R\big) &= \mathbf{Q}^R = \text{RoPE}\big(\mathbf{C}^Q\boldsymbol{W}^{QR}\big), \\
\mathbf{Q} &= \text{Concat}\big(\mathbf{Q}^C, \mathbf{Q}^R\big).
\end{aligned}$$

The weight matrices are $\boldsymbol{W}^{DQ} \in \mathbb{R}^{d_{\text{model}} \times d_c'}$, $\boldsymbol{W}^{UQ} \in \mathbb{R}^{d_c' \times (d_h h)}$, and $\boldsymbol{W}^{QR} \in \mathbb{R}^{d_c' \times (d_h^R h)}$. Here, $\mathbf{C}^Q \in \mathbb{R}^{T \times d_c'}$ (where $d_c' \ll d_h h$) is the compressed query latent. The final query $\mathbf{Q}_i$ for each head, formed by concatenating $\mathbf{Q}_i^C$ and $\mathbf{Q}_i^R$, has a dimension of $d_h + d_h^R$.

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

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

where $\mathbf{V}_i$ is typically $\mathbf{V}_i^C$ as no residual value component is explicitly defined, and $\boldsymbol{W}^O \in \mathbb{R}^{(d_h h) \times d_{\text{model}}}$ is the output projection.

During inference, $\mathbf{C}^{KV}$ and $\mathbf{K}^R$ are cached to accelerate decoding. In detail, if RoPE were ignored for the compressed components, the inner product $\mathbf{q}_{t,i}^\top \mathbf{k}_{s,i}$ (where $\mathbf{q}_{t,i}, \mathbf{k}_{s,i} \in \mathbb{R}^{d_h}$) of the i-th head between t-th token query and s-th token key could be calculated using the current hidden state $\mathbf{x}_t \in \mathbb{R}^{d_{\text{model}}}$ and the cached latent state $\mathbf{c}_s^{KV} \in \mathbb{R}^{d_c}$ for the s-th token:

$$\begin{aligned}
\mathbf{q}_{t,i}^\top \mathbf{k}_{s,i} &= [(\boldsymbol{W}_i^{UQ})^\top (\boldsymbol{W}_i^{DQ})^\top \mathbf{x}_t]^\top [(\boldsymbol{W}_i^{UK})^\top \mathbf{c}_s^{KV}] && \text{(H.1)} \\
&= \mathbf{x}_t^\top [\boldsymbol{W}_i^{DQ}\boldsymbol{W}_i^{UQ}(\boldsymbol{W}_i^{UK})^\top]\mathbf{c}_s^{KV}, && \text{(H.2)}
\end{aligned}$$

where $\boldsymbol{W}_i^{(\cdot)}$ denotes the i-th head's portion of the respective weight matrix.
The term $[\boldsymbol{W}_i^{DQ}\boldsymbol{W}_i^{UQ}(\boldsymbol{W}_i^{UK})^\top]$ could be pre-computed for faster decoding. However, as noted

by [50], this pre-computation strategy is not directly compatible with RoPE if RoPE were applied to these compressed representations. RoPE applies a rotation matrix $\mathbf{T}_t \in \mathbb{R}^{d_h \times d_h}$ based on position t (see Section H.5), satisfying $\mathbf{T}_t \mathbf{T}_s^\top = \mathbf{T}_{t-s}$ (Equation H.4). If RoPE were applied to the up-projected Q^C and K^C:

$$\begin{aligned} \mathbf{q}_{t,i}^\top \mathbf{k}_{s,i} &= [\mathbf{T}_t{}^\top (\boldsymbol{W}_i^{UQ})^\top (\boldsymbol{W}_i^{DQ})^\top \mathbf{x}_t]^\top [\mathbf{T}_s{}^\top (\boldsymbol{W}_i^{UK})^\top \mathbf{c}_s^{KV}] \\ &= \mathbf{x}_t^\top [\boldsymbol{W}_i^{DQ} \boldsymbol{W}_i^{UQ} \mathbf{T}_{t-s} (\boldsymbol{W}_i^{UK})^\top] \mathbf{c}_s^{KV}. \end{aligned} \tag{H.3}$$

Unlike Equation (H.2), acceleration by pre-computing the term $[\boldsymbol{W}_i^{DQ} \boldsymbol{W}_i^{UQ} \mathbf{T}_{t-s} (\boldsymbol{W}_i^{UK})^\top]$ is not possible because it depends on the relative position $(t-s)$ and thus varies for different (t, s) pairs. To maintain RoPE compatibility while benefiting from compression, MLA introduces an additional, smaller key component $\mathbf{K}^R$ (and similarly $\mathbf{Q}^R$) to which RoPE is applied, while the main compressed components $\mathbf{K}^C$ and $\mathbf{V}^C$ (derived from $\mathbf{C}^{KV}$) remain RoPE-free. As we will demonstrate in Section 3.2 of the main paper, TPA offers a different approach to integrate RoPE efficiently with factorized attention through its tensor product formulation.

H.4 Multi-matrix Factorization Attention (MFA)

[20] proposed Multi-matrix Factorization Attention (MFA), which can be conceptualized as a variation of MQA where the shared key and value projections have a dimension d_c, and the query projection for each head is low-rank factorized:

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

where

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

H.5 Rotary Position Embedding (RoPE)

Many recent LLMs use rotary position embedding (RoPE; 51) to encode positional information in the query/key vectors. Specifically, for a vector at position t, RoPE applies a rotation matrix $\mathbf{T}_t \in \mathbb{R}^{d \times d}$ (where d is the dimension of the query/key vectors, typically d_h per head). $\mathbf{T}_t$ is a block-diagonal matrix composed of $d/2$ rotation blocks of the form $\begin{pmatrix} \cos(t\theta_j) & -\sin(t\theta_j) \\ \sin(t\theta_j) & \cos(t\theta_j) \end{pmatrix}$ for $j \in \{1, \ldots, d/2\}$. The frequencies $\{\theta_j\}$ are typically defined as $\theta_j = \text{base}^{-2j/d}$, with a common base like 10000. If $\mathbf{q}_t \in \mathbb{R}^d$ is a query (or key) row vector for a specific head at position t, RoPE is applied as:

$$\text{RoPE}(\mathbf{q}_t) \triangleq \mathbf{q}_t \mathbf{T}_t.$$

A key property of RoPE is that the inner product between RoPE-transformed vectors depends only on their relative position. For a query $\mathbf{q}_t$ and key $\mathbf{k}_s$: $(\mathbf{q}_t \mathbf{T}_t)(\mathbf{k}_s \mathbf{T}_s)^\top = \mathbf{q}_t \mathbf{T}_t \mathbf{T}_s^\top \mathbf{k}_s^\top = \mathbf{q}_t \mathbf{T}_{t-s} \mathbf{k}_s^\top$. This relies on the property:

$$\mathbf{T}_t \mathbf{T}_s^\top = \mathbf{T}_{t-s}, \tag{H.4}$$

which embeds relative positional information $(t-s)$ into the attention scores.

I More on TPA

Parameter Initialization for TPA Factors. We initialize the weight matrices for TPA factors, such as $\boldsymbol{W}_r^{a^Q}$, $\boldsymbol{W}_r^{a^K}$, $\boldsymbol{W}_r^{a^V}$, $\boldsymbol{W}_r^{b^Q}$, $\boldsymbol{W}_r^{b^K}$, and $\boldsymbol{W}_r^{b^V}$ (or their combined forms $\boldsymbol{W}^{a^Q}$, $\boldsymbol{W}^{b^Q}$, etc.), using Xavier initialization [14]. Specifically, each entry of a weight matrix is drawn from a uniform distribution $\mathcal{U}(-bound, bound)$, where $bound = \sqrt{6/(n_{\text{in}} + n_{\text{out}})}$. Here, n_{in} and n_{out} are the input and output dimensions of the respective weight matrix. This initialization strategy is chosen to help maintain the variance of activations and gradients as they propagate through the network layers, contributing to stable training.

TPA with Non-contextual B. In Section A.1, we have introduced TPA with non-contextual A, where head-dimension factors $\mathbf{a}_r^Q, \mathbf{a}_r^K, \mathbf{a}_r^V \in \mathbb{R}^h$ are fixed. 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 the input token $\mathbf{x}_t$. The key tensor for token t, $\mathbf{K}_t \in \mathbb{R}^{h \times d_h}$, would then be constructed as:

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

A similar formulation applies to values. This configuration might be effective if the fundamental token-level features (captured by $\mathbf{b}_r$) are relatively stable, while their combination across heads (captured by $\mathbf{a}_r(\mathbf{x}_t)$) needs to adapt to the context. Performance comparisons for TPA with non-contextual A factors versus non-contextual B factors on small and medium-sized models are presented in Tables 5, 6, 7, and 8.

Table 5: Evaluation results of small models with TPA using non-contextual A or B factors, pretrained on FineWeb-Edu 100B dataset (0-shot with lm-evaluation-harness). Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
TPA (non-ctx-A)	50.17	25.60	57.95	36.13	31.40	64.80	49.57	24.88	64.80	45.03
TPA (non-ctx-B)	47.39	26.37	54.8	32.71	30.2	63.38	50.2	23.13	64.8	43.66

Table 6: Evaluation results of small models with TPA using non-contextual A or B factors, pretrained on FineWeb-Edu 100B dataset (2-shot with lm-evaluation-harness). Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
TPA (non-ctx-A)	55.09	27.65	53.82	36.24	30.20	64.53	50.75	26.01	78.60	46.99
TPA (non-ctx-B)	50.8	26.96	57.65	32.4	29.4	63.22	49.57	23.96	66.4	44.48

TPA KV Only. A simpler variant involves using a standard linear projection for queries,

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

and factorize only the key and value tensors ($\mathbf{K}_t, \mathbf{V}_t$). This approach, termed TPA-KVonly, maintains the standard query projection mechanism but still achieves significant KV cache reduction through factorized key and value representations.

Table 7: Evaluation results of medium models with TPA using non-contextual A or B factors, pre-trained on FineWeb-Edu 100B dataset (0-shot with lm-evaluation-harness). Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
TPA (non-ctx-A)	58.96	31.48	59.76	45.07	34.80	69.21	53.59	25.42	76.40	50.52
TPA (non-ctx-B)	55.43	29.69	58.32	40.77	34.40	66.92	51.38	25.66	71.10	48.19

Table 8: Evaluation results of medium models with TPA using non-contextual A or B factors, pre-trained on FineWeb-Edu 100B dataset (2-shot with lm-evaluation-harness). Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
TPA (non-ctx-A)	65.45	33.79	56.88	45.23	33.60	68.61	54.22	25.00	85.00	51.98
TPA (non-ctx-B)	61.20	30.20	55.93	40.45	34.40	68.23	51.78	26.11	78.10	49.60

TPA KV with Shared B. Further parameter reduction can be achieved by sharing the token-dimension factors $\mathbf{b}_r$ between keys and values:

$$\mathbf{b}_r^K(\mathbf{x}_t) = \mathbf{b}_r^V(\mathbf{x}_t) \quad \text{(if contextual)}, \quad \text{or} \quad \mathbf{b}_r^K = \mathbf{b}_r^V \quad \text{(if non-contextual)}.$$

This sharing reduces both parameter count and the KV cache footprint. Although it constrains $\mathbf{K}_t$ and $\mathbf{V}_t$ to be constructed from the same token-level basis vectors, this variant can still offer strong performance with additional memory savings.

Nonlinear Head Factors. Instead of using purely linear transformations to derive the contextual head-dimension factors $\mathbf{a}_r^Q(\mathbf{x}_t), \mathbf{a}_r^K(\mathbf{x}_t), \mathbf{a}_r^V(\mathbf{x}_t)$, one can introduce element-wise nonlinearities (e.g., sigmoid $\sigma(\cdot)$ or softmax). Applying softmax, for instance, to the coefficients that generate $\mathbf{a}_r(\mathbf{x}_t)$ could be interpreted as a form of Mixture-of-Heads, where the network learns to dynamically weight different head configurations based on the input context.

Discussion. These variants highlight the flexibility of the TPA framework, allowing for different trade-offs between memory efficiency, computational cost, and model expressiveness. By carefully choosing which factor components (head-dimension or token-dimension) are contextual versus non-contextual, and by adjusting the ranks (R_Q, R_K, R_V), TPA can not only unify existing mechanisms like MHA, MQA, and GQA but also significantly reduce KV cache size—potentially by an order of magnitude—during autoregressive inference.

J More on Experiments

J.1 Experimental Settings

We list the main architecture hyper-parameters and training devices in Table 9. For all models, the head dimension d_h is fixed at 64. Specific architectural choices include: 2 KV heads for GQA models; a residual key dimension $d_h^R = 32$ for MLA models; and ranks $R_K = R_V = 2$ and $R_Q = 6$ for TPA and TPA-KVonly models, unless otherwise specified. Other relevant hyper-parameters are listed in Table 10.

Training Setup Details. We follow the `nanoGPT` training configuration [23]. In particular, we use the AdamW [34] 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 [35] with $2{,}000$ warmup steps and a (total) global batch size of 480. For the *small*, *medium*, *large* and *XL* models, we set maximum learning rates of 6×10^{-4}, 3×10^{-4}, 2×10^{-4}, and 1×10^{-4} (respectively), and minimum learning rates of 3×10^{-5}, 6×10^{-5}, 1×10^{-5}, and 1×10^{-5} (respectively).

Table 9: The architecture hyper-parameters and training devices of models. Abbreviations: BS. = Batch Size, GAS. = Gradient Accumulation Steps.

MODEL SIZE	PARAMETERS	DEVICES	MICRO BS.	GAS.	#LAYERS	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
XL	1.55B	8× A100 GPUS	6	10	48	1600

Table 10: The architecture hyper-parameters for different models.

MODEL SIZE	SMALL	MEDIUM	LARGE	XL
h (MHA)	12	16	20	25
h (MQA)	23	31	39	49
h (GQA)	22	30	38	48
h (MLA)	12	23	34	49
h (TPA-KVONLY)	22	29	37	47
h (TPA)	34	47	61	78
d_c (MLA)	256	512	512	512
d'_c (MLA)	512	1024	1024	1024

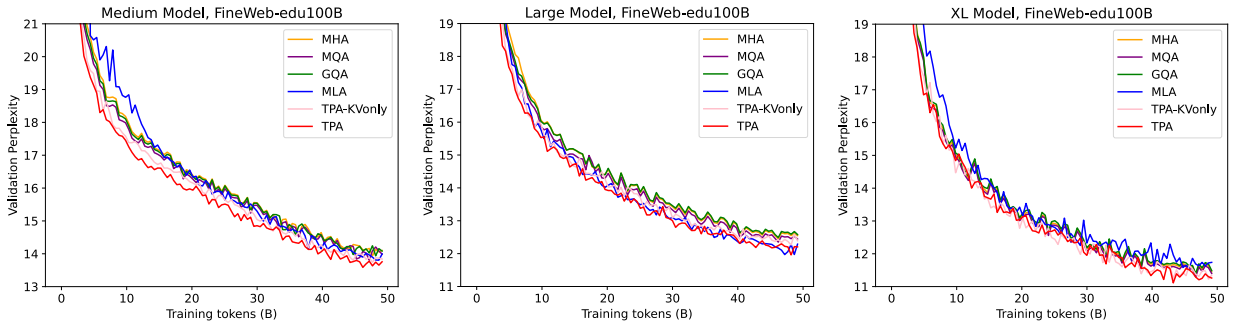
J.2 Additional Experimental Results

J.2.1 Perplexity Curves

We display the perplexity curves for medium, large, and XL size models in Figure 9.

J.2.2 Ablation Study on Different Ranks

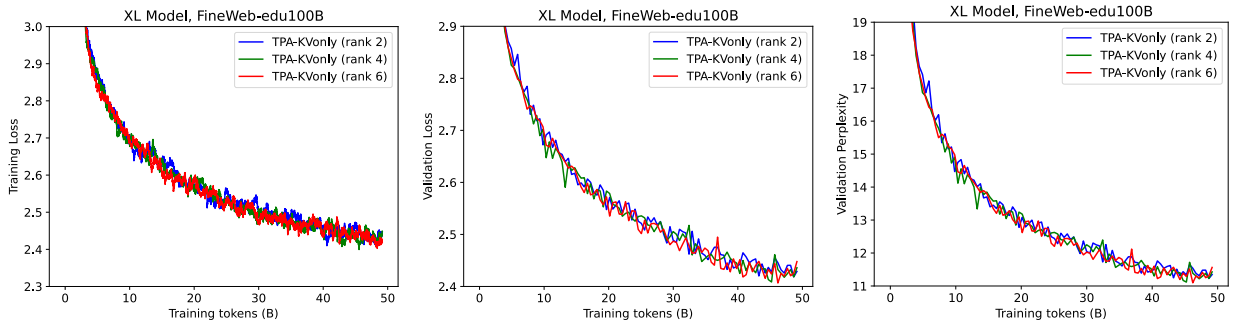
Figure 10 illustrates the training loss, validation loss, and validation perplexity for XL-sized (1.5B parameters) TPA models with varying key/value ranks ($R_K = R_V = R$, as indicated in the figure legend), trained on the FineWeb-Edu 100B dataset. Corresponding 0-shot evaluation results are presented in Table 12 (rows for TPA-KVonly with different $R_{K,V}$). These results indicate that



(a) Validation Perplexity (b) Validation Perplexity (c) Validation Perplexity

Figure 9: The validation perplexity of medium-size (353M) models, large-size (773M), and XL-size (1.5B) models with different attention mechanisms on the FineWeb-Edu 100B dataset.

increasing the ranks for key and value factorizations generally improves the performance of the TPA models.



(a) Training Loss (b) Validation Loss (c) Validation Perplexity

Figure 10: The training loss, validation loss and validation perplexity curves of XL-size (1.5B) TPA models with different key/value ranks ($R_K = R_V = R$) on the FineWeb-Edu 100B dataset.

J.2.3 0-shot Evaluation with lm-evaluation-harness

We present 0-shot evaluation results using the lm-evaluation-harness for small (124M parameters) and XL (1.5B parameters) models in Tables 11 and 12, respectively.

J.2.4 2-shot Evaluation with lm-evaluation-harness

Similarly, 2-shot evaluation results are provided in Tables 13 (Small), 14 (Medium), 15 (Large), and 16 (XL).

J.3 Ablation Studies on Learning Rates

To assess sensitivity to learning rates, we conducted parallel experiments on medium-sized models using a learning rate of 3×10^{-4} (compared to the default 6×10^{-4} used for other medium model

Table 11: Evaluation results of small models (124M) with different attention mechanisms, pre-trained on 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	50.21	26.71	58.01	36.25	**32.80**	64.69	50.59	**24.67**	71.90	46.20
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**

Table 12: Evaluation results of XL models (1.5B) with different attention mechanisms, pre-trained on 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. If not specified, TPA and TPA-KVonly models use $R_K = R_V = 2$.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	64.81	35.41	61.90	54.32	37.20	72.74	55.80	25.44	**82.80**	54.49
MQA	64.10	36.01	62.26	54.38	39.00	72.58	56.43	23.70	81.90	54.48
GQA	63.68	35.92	60.46	54.17	38.40	**73.56**	56.27	24.77	81.70	54.33
MLA	64.14	35.92	60.12	53.60	39.20	72.25	55.17	24.71	81.60	54.08
TPA-KVonly	65.61	36.77	**63.02**	54.17	37.00	73.34	54.62	25.02	81.60	54.57
TPA-KVonly ($R_{K,V} = 4$)	64.52	**37.03**	63.27	**54.89**	39.80	72.91	56.51	24.74	81.60	**55.03**
TPA-KVonly ($R_{K,V} = 6$)	65.78	35.92	61.71	54.86	38.60	72.69	**57.93**	**25.59**	82.20	**55.03**
TPA	**66.71**	36.52	61.38	54.03	**40.40**	72.52	56.83	24.49	82.20	55.01

Table 13: Evaluation results of small models (124M) with different attention mechanisms, pre-trained 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	54.76	27.13	**58.07**	36.13	31.40	65.07	51.30	25.90	78.90	47.63
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

results). The training loss, validation loss, and validation perplexity curves are shown in Figure 11. Performance on standard benchmarks for these models trained with the 3×10^{-4} learning rate are reported in Tables 17 (0-shot) and 18 (2-shot). The results demonstrate that TPA and TPA-KVonly maintain their performance advantages over other attention mechanisms even with this alternative learning rate.

Table 14: Evaluation results of medium models (353M) with different attention mechanisms, pre-trained on FineWeb-Edu 100B dataset (2-shot with lm-evaluation-harness, default LR 6×10^{-4}). The best scores in each column are **bolded**. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

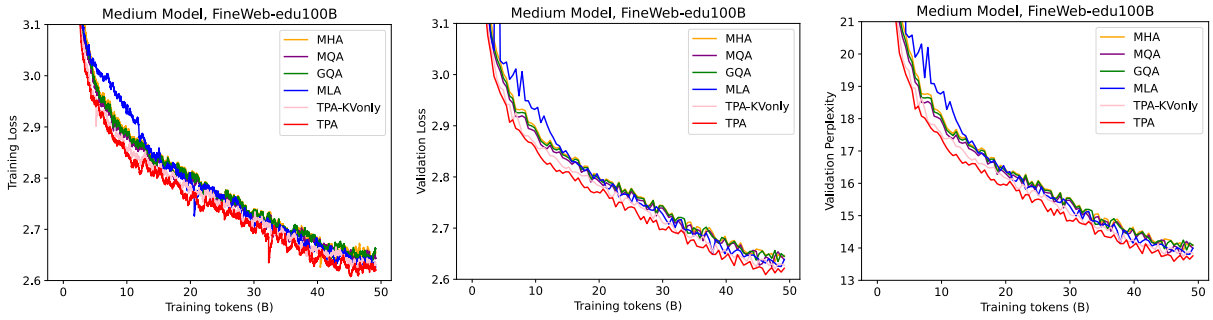
Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	64.73	32.42	58.29	45.89	34.20	68.50	53.20	**25.86**	88.00	52.34
MQA	64.98	33.62	55.02	45.81	34.00	69.59	53.43	24.30	85.20	51.77
GQA	65.24	33.19	56.54	45.41	34.80	69.04	**55.72**	24.73	87.90	52.51
MLA	64.98	33.62	53.52	45.94	33.00	68.55	51.85	25.46	89.10	51.78
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**

Table 15: Evaluation results of large models (772M) with different attention mechanisms, pre-trained on 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	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	70.54	**38.74**	**61.50**	**51.86**	36.00	70.89	54.22	25.47	**92.40**	**55.74**
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

Table 16: Evaluation results of XL models (1.5B) with different attention mechanisms, pre-trained on 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. If not specified, $R_K = R_V = 2$ for TPA and TPA-KVonly models.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	70.83	39.93	59.85	54.05	36.20	72.52	55.17	25.42	91.70	56.18
MQA	71.34	39.76	58.93	54.27	39.40	72.96	57.38	24.74	91.90	56.74
GQA	71.17	39.08	60.18	54.05	37.40	73.07	56.35	24.87	**92.20**	56.49
MLA	70.79	37.54	50.83	53.33	**40.00**	72.09	56.51	24.93	91.80	55.31
TPA-KVonly	72.85	39.68	60.92	53.81	37.00	**73.34**	56.83	**26.19**	91.30	56.88
TPA-KVonly ($R_{K,V} = 4$)	72.98	**40.27**	60.15	**54.88**	36.80	73.29	56.43	25.50	92.10	56.93
TPA-KVonly ($R_{K,V} = 6$)	**73.95**	39.76	58.99	54.73	36.80	72.91	**59.04**	24.93	92.90	**57.11**
TPA	71.76	39.16	**61.25**	53.74	37.80	72.80	55.49	23.86	90.70	56.28



(a) Training Loss (b) Validation Loss (c) Validation Perplexity

Figure 11: The training loss, validation loss, and validation perplexity of medium-size (353M) models (learning rate 3×10^{-4}) with different attention mechanisms on the FineWeb-Edu 100B dataset.

Table 17: The evaluation results of medium models (learning rate 3×10^{-4}) 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	**59.64**	29.78	60.73	45.17	34.20	68.66	52.80	25.34	75.70	50.22
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**

Table 18: The evaluation results of medium models (learning rate 3×10^{-4}) 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	51.42
GQA	61.70	32.17	52.81	43.99	33.80	68.50	53.35	24.44	86.40	50.80
MLA	65.95	31.48	50.98	44.99	32.20	68.93	51.93	25.89	88.80	51.24
TPA-KVonly	65.99	33.70	57.49	44.47	**34.20**	**69.53**	53.28	24.23	86.50	52.15
TPA	**66.54**	**34.47**	58.96	**45.35**	33.00	69.21	**53.99**	24.51	**91.30**	**53.04**