Flash Tensor Product Attention

TPA Authors

Abstract

Flash Tensor Product Attention.

1 Toward Faster Computation Without Materializing Q, K and V

We now explore whether it is possible to compute the attention scores $\mathbf{Q}\,\mathbf{K}^\top$ of Tensor Product Attention (TPA) (Zhang et al., 2025) *directly* from their factorized forms, thereby reducing floating-point operations.

1.1 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):

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

and each key vector $\mathbf{K}_\tau^{(i)} \in \mathbb{R}^{d_h}$ is factorized (with rank R_k):

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

Their dot-product for tokens t, τ is

$$\big[\mathbf{Q}^{(i)}\,(\mathbf{K}^{(i)})^\top\big]_{t,\tau} \;=\; \frac{1}{R_q R_k} \sum_{r=1}^{R_q} \sum_{s=1}^{R_k} a_{q,i}^{(r)}(\mathbf{x}_t)\, a_{k,i}^{(s)}(\mathbf{x}_\tau)\, \big\langle \mathbf{b}_q^{(r)}(\mathbf{x}_t), \mathbf{b}_k^{(s)}(\mathbf{x}_\tau)\big\rangle. \tag{1.1}$$

1.2 Multi-Head Case

For multi-head attention with h heads, one repeats the factorization across all heads. The $\mathbf{b}_q^{(r)}, \mathbf{b}_k^{(s)}$ vectors are shared across heads.

1.3 Complexity Analysis

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

1. **Naïve:** Materialize $\mathbf{Q}$ and $\mathbf{K}$ from factors, then perform the usual batched GEMM.
2. **Specialized:** Attempt to compute $\mathbf{Q}\,\mathbf{K}^\top$ directly from the rank-(R_q, R_k) factors without explicitly forming $\mathbf{Q}, \mathbf{K}$.

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* $\mathbf{Q},\mathbf{K}$*:* $\mathcal{O}\big(B\,T\,(R_q\,h\,d_h+R_k\,h\,d_h)\big)$.
- *Dot-product* $\mathbf{Q}\,(\mathbf{K})^\top$*:* $\mathcal{O}\big(B\,T^2\,d_{\text{model}}\big)$.

Typically $R_q, R_k, R_v \ll h$, so the overhead of constructing factors is small relative to $\mathcal{O}(T^2\,d_{\text{model}})$. Meanwhile, we still gain KV caching benefits.

TPA: Specialized Implementation. If we bypass explicitly forming $\mathbf{Q},\mathbf{K}$, each dot product $\mathbf{Q}_t\cdot\mathbf{K}_\tau$ is a double sum over rank indices. Below we detail its complexity.

1.4 Complexity Analysis for the Specialized Implementation

Single-Head Complexity. A single attention head of dimension d_h. For each query:

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

and for each key:

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

Their dot product:

$$\mathbf{Q}_t^{(i)}\cdot\mathbf{K}_\tau^{(i)} \;=\; \frac{1}{R_qR_k}\sum_{r=1}^{R_q}\sum_{s=1}^{R_k}\Big[a_{q,i}^{(r)}(\mathbf{x}_t)\,a_{k,i}^{(s)}(\mathbf{x}_\tau)\Big]\,\big\langle\mathbf{b}_q^{(r)}(\mathbf{x}_t),\mathbf{b}_k^{(s)}(\mathbf{x}_\tau)\big\rangle.$$

For each pair (r,s), we pay:

1. $\mathcal{O}(1)$ for multiplying two scalars,
2. $\mathcal{O}(d_h)$ for the dot product $\mathbf{b}_q^{(r)}(\mathbf{x}_t)\cdot\mathbf{b}_k^{(s)}(\mathbf{x}_\tau)$.

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 T queries and T keys, that is $\mathcal{O}(T^2\,R_q\,R_k\,d_h)$ for a single head.

Multi-Head and Batches (Reusing $\mathbf{b}$-Dot Products). When extending to h heads, each head i has its own scalar factors $\mathbf{a}_{q,i}^{(r)}(\mathbf{x}_t)$ and $\mathbf{a}_{k,i}^{(s)}(\mathbf{x}_\tau)$, but the $\mathbf{b}$-vectors $\mathbf{b}_q^{(r)}(\mathbf{x}_t)$ and $\mathbf{b}_k^{(s)}(\mathbf{x}_\tau)$ can still be *shared* across all heads (assuming the same rank-R factors for every head). Hence, one can split the total cost into two stages:

1. **$\mathbf{b}$-Dot-Product Stage:**
   For each token pair (t,τ) and each rank pair (r,s), compute the dot product

   $$\big\langle\mathbf{b}_q^{(r)}(\mathbf{x}_t),\;\mathbf{b}_k^{(s)}(\mathbf{x}_\tau)\big\rangle\;\in\;\mathbb{R}.$$

   Since each dot product is $\mathcal{O}(d_h)$ and there are R_qR_k rank pairs as well as T^2 token pairs, this stage costs:

   $$\mathcal{O}\big(T^2\,R_qR_k\,d_h\big).$$

   Crucially, these $\mathbf{b}$-dot products need only be computed *once* and can be cached for reuse by all heads.

2. **Per-Head Scalar Multiplications:**
After the $\mathbf{b}$-dot products are precomputed (and cached), each head i only needs to multiply each stored dot product by the corresponding scalars $\mathbf{a}_{q,i}^{(r)}(\mathbf{x}_t)\,\mathbf{a}_{k,i}^{(s)}(\mathbf{x}_\tau)$. Since this scalar multiplication is $\mathcal{O}(1)$ per pair, and there are T^2 token pairs and $R_q R_k$ rank pairs for each of the h heads, this step costs:
$$\mathcal{O}\big(h\,T^2\,R_q R_k\big).$$

Putting these together, for batch size B, the total cost is

$$\mathcal{O}\big(B\,T^2\,R_q R_k\,d_h\big) \;+\; \mathcal{O}\big(B\,T^2\,h\,R_q R_k\big) \;=\; \mathcal{O}\Big(B\,T^2\,R_q R_k\big(d_h + h\big)\Big).$$

By contrast, the standard multi-head attention dot-product step is $\mathcal{O}\big(B\,T^2\,h\,d_h\big)$. Hence, for the specialized TPA approach to *reduce* flops,

$$R_q R_k\,(d_h + h) \;\leq\; h\,d_h.$$

Thus a practical guideline is to ensure $R_q R_k \;<\; h\,\frac{d_h}{d_h+h}$. When that holds, bypassing explicit materialization of $\mathbf{Q}$ and $\mathbf{K}$ can be beneficial.

1.5 Toward Faster Computation Without Materializing Q, K, V

We have explored a two-step procedure for computing $\mathbf{Q}\,\mathbf{K}^\top$ directly from factorized queries and keys *without* materializing $\mathbf{Q}$ or $\mathbf{K}$. Here, we extend this idea to also avoid explicitly forming $\mathbf{V}$. That is, all three activations $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ remain factorized throughout the attention pipeline. We present a single-head formulation below, and then discuss multi-head and batch extensions.

Extending the Two-Step Approach to Avoid V Materialization. After we obtain $\mathbf{Q}\mathbf{K}^\top$, we apply $\alpha_{t,\tau} = \mathrm{softmax}\big(\frac{1}{\sqrt{d_h}}\,(QK^\top)_{t,\tau}\big)$. The final attention output at token t (single head) is

$$\mathrm{head}(t) \;=\; \sum_{\tau=1}^{T} \alpha_{t,\tau}\,\mathbf{V}_\tau.$$

Using the factorization $\mathbf{V}_\tau = \frac{1}{R_v}\sum_{u=1}^{R_v} a_v^{(u)}(\mathbf{x}_\tau)\,\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)$, we write:

$$\mathrm{head}(t) \;=\; \frac{1}{R_v}\sum_{\tau=1}^{T}\alpha_{t,\tau}\sum_{u=1}^{R_v} a_v^{(u)}(\mathbf{x}_\tau)\,\mathbf{b}_v^{(u)}(\mathbf{x}_\tau).$$

Rearrange sums:

$$\mathrm{head}(t) \;=\; \frac{1}{R_v}\sum_{u=1}^{R_v}\Big[\sum_{\tau=1}^{T}\big(\alpha_{t,\tau}\,a_v^{(u)}(\mathbf{x}_\tau)\big)\,\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)\Big].$$

We *still* do not explicitly form $\mathbf{V}_\tau$. Instead:

Stage 1: Calculating $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)$ for all tokens. We simply observe that each output $\mathrm{head}(t)$ can be computed by summing vectors $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau) \in \mathbb{R}^{d_h}$ weighted by $\alpha_{t,\tau}\,a_v^{(u)}(\mathbf{x}_\tau)$. The complexity for constructing $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)\;\forall u,\tau$ is $\mathcal{O}(T\,R_v\,d_h)$.

Stage 2: Weighted Summation by $\alpha_{t,\tau}\,a_v^{(u)}(\mathbf{x}_\tau)$. For each token t, the final attention head output is

$$\frac{1}{R_v}\sum_{\tau=1}^{T}\alpha_{t,\tau}\sum_{u=1}^{R_v} a_v^{(u)}(\mathbf{x}_\tau)\,\mathbf{b}_v^{(u)}(\mathbf{x}_\tau) \;=\; \frac{1}{R_v}\sum_{u=1}^{R_v}\Big[\sum_{\tau=1}^{T}\big(\alpha_{t,\tau}\,a_v^{(u)}(\mathbf{x}_\tau)\big)\,\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)\Big].$$

We still never explicitly materialize $\mathbf{V}$. Instead, for each pair (t,u), we must accumulate the sum of T vectors $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau) \in \mathbb{R}^{d_h}$, each scaled by the scalar $\alpha_{t,\tau}\,a_v^{(u)}(\mathbf{x}_\tau)$. Because each vector is d_h-dimensional, each (t,u) summation costs $\mathcal{O}(T\,d_h)$. Summed over $t = 1\ldots T$ and $u = 1\ldots R_v$, the total work is $\mathcal{O}(T^2\,R_v\,d_h)$ for the entire sequence.

In practice, one precomputes all $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)$ for $\tau = 1\ldots N$, so each accumulation can be implemented as a simple "scalar-times-vector add" in a tight loop. This cost is usually smaller than the $\mathbf{Q}\mathbf{K}^\top$ factorized cost if $R_v \ll h$.

1.6 Overall Complexity for Single-Head

Combining the four bullet-point stages from above (ignoring smaller overheads like the softmax) yields:

(i) **QK $\mathbf{b}$-Dot Product Stage:** $\mathcal{O}(T^2\, R_q\, R_k\, d_h)$.

(ii) **QK Scalar-Multiply Stage:** $\mathcal{O}(T^2\, R_q\, R_k)$.

(iii) **Computing $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)$ for all tokens:** $\mathcal{O}(T\, R_v\, d_h)$.

(iv) **Weighted Summation by $\alpha_{t,\tau}\, a_v^{(u)}(\mathbf{x}_\tau)$:** $\mathcal{O}(T^2\, R_v\, d_h)$.

Hence, for a single head, the total cost is:

$$\mathcal{O}\Big(T^2\, R_q\, R_k\, d_h \;+\; T^2\, R_q\, R_k \;+\; T\, R_v\, d_h \;+\; T^2\, R_v\, d_h\Big).$$

In many cases (especially for large T), the $\mathcal{O}(T^2)$ terms dominate, so one often focuses on

$$\mathcal{O}\Big(T^2\, R_q\, R_k\, d_h \;+\; T^2\, R_q\, R_k \;+\; T^2\, R_v\, d_h\Big).$$

1.7 Multi-Head and Batch Extensions (Reuse of $\mathbf{b}$-Dot Products)

When extending to h heads and batch size B, all sequence-length-dependent terms are multiplied by $\sim B\, h$. However, crucial $\mathbf{b}$-dot products can be shared across heads:

QK $\mathbf{b}$-Dot Products. Since each head has distinct scalar factors $a_{q,i},\, a_{k,i}$ but the same $\mathbf{b}_q^{(r)},\, \mathbf{b}_k^{(s)}$ across heads, each pairwise dot product

$$\langle \mathbf{b}_q^{(r)}(\mathbf{x}_t),\, \mathbf{b}_k^{(s)}(\mathbf{x}_\tau) \rangle$$

is computed just once per batch. That cost remains

$$\mathcal{O}\big(B\, T^2\, R_q\, R_k\, d_h\big),$$

not multiplied by h. After caching these dot products, each of the h heads pays $\mathcal{O}\big(B\, T^2\, h\, R_q\, R_k\big)$ total for the head-specific scalar multiplications (the "$\alpha_{t,\tau}$"–like factors).

V $\mathbf{b}$-Evaluations. Likewise, the $\mathbf{b}_v^{(u)}$ factors are shared across heads (i.e. one set of $\mathbf{b}_v$-vectors for all heads). Hence, computing all $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)$ for $\tau = 1 \ldots T$ (across the batch) is a one-time cost:

$$\mathcal{O}\big(B\, T\, R_v\, d_h\big).$$

Then each head i has its own scalar factors $a_{v,i}^{(u)}(\mathbf{x}_\tau)$, so the final accumulation $\sum_{\tau=1}^{T} \alpha_{t,\tau}\, a_{v,i}^{(u)}(\mathbf{x}_\tau)\, \mathbf{b}_v^{(u)}(\mathbf{x}_\tau)$ costs $\mathcal{O}\big(B\, T^2\, h\, R_v\, d_h\big)$ in total (for all t, u).

Putting it all together, the total flops for multi-head attention with batch size B are:

$$\underbrace{\mathcal{O}\big(B\, T^2\, R_q\, R_k\, d_h\big)}_{\substack{\text{QK } \mathbf{b}\text{-dot products} \\ \text{(shared across heads)}}} + \underbrace{\mathcal{O}\big(B\, T^2\, h\, R_q\, R_k\big)}_{\text{per-head QK scalar mult.}} + \underbrace{\mathcal{O}\big(B\, T\, R_v\, d_h\big)}_{\substack{\text{Compute } \mathbf{b}_v \text{ for all tokens} \\ \text{(shared across heads)}}} + \underbrace{\mathcal{O}\big(B\, T^2\, h\, R_v\, d_h\big)}_{\substack{\text{final accumulations} \\ \text{(per head)}}}.$$

Discussion. By contrast, standard multi-head attention typically requires $\mathcal{O}\big(B\, T^2\, h\, d_h\big)$ flops for the $\mathbf{Q}\mathbf{K}^\top$ dot product (plus a similar $\mathcal{O}\big(B\, T^2\, h\, d_h\big)$ for multiplying by $\mathbf{V}$). The factorization can yield savings provided $R_q R_k \ll h$ (for QK) and $R_v \ll h$ (for V), though actual speedups depend on how well these multi-stage kernels are implemented and on hardware efficiency. By retaining $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ in factorized form, one can forgo the usual steps:

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

Instead, the large $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors (of size $T \times d_h$) are never materialized. The cost is replaced by rank-based $\mathbf{b}$-dot-product computations plus per-head scalar multiplications. The main challenge is to keep the factor ranks (R_q, R_k, R_v) sufficiently small relative to d_h and to implement the necessary multi-stage kernels efficiently. When $R_q, R_k, R_v \ll h$, fully factorized QKV attention can yield substantial gains in both computation and memory footprint.

1.8 Decoding Speed during Inference Time of MHA, MQA, GQA, MLA, and TPA

Suppose we are in an autoregressive setting, decoding the current token $\mathbf{x}_T$ given cached keys and values (KV) from all previous tokens $\mathbf{x}_1, \ldots, \mathbf{x}_{T-1}$. For each attention head $i \in \{1, \ldots, h\}$, we store $\mathbf{K}_i \in \mathbb{R}^{T \times d_h}, \mathbf{V}_i \in \mathbb{R}^{T \times d_h}$. Below, we compare the flops needed by MHA, MQA, GQA, MLA, and TPA to compute the next-token logits during inference.

MHA, MQA, and GQA. Despite sharing or grouping keys/values in MQA and GQA, the *decoding* cost for MHA, MQA, and GQA remains of the same order. Specifically, for each head i, we compute:

$$\mathbf{Q}_i(\mathbf{x}_T) \in \mathbb{R}^{d_h}, \quad \mathbf{K}_i \in \mathbb{R}^{T \times d_h}, \quad \mathbf{Q}_i(\mathbf{x}_T)\,\mathbf{K}_i^\top \in \mathbb{R}^{1 \times T},$$
$$\text{and} \quad \mathrm{Softmax}\big(\mathbf{Q}_i(\mathbf{x}_T)\,\mathbf{K}_i^\top\big)\,\mathbf{V}_i \in \mathbb{R}^{d_h}.$$

Hence, the flops scale linearly in h, d_h, and T. For example, forming $\mathbf{Q}_i(\mathbf{x}_T)\mathbf{K}_i^\top$ for each head i costs roughly $\mathcal{O}(h\, d_h\, T)$.

MLA. During inference, MLA can be seen as MQA but uses a larger head dimension to accommodate both RoPE and compressed representations (e.g., $d_h' = d_{\text{rope}} + d_c$). In typical configurations, $d_{\text{rope}} + d_c$ can be significantly larger (e.g., $d_h' = 576$ rather than $d_h = 64$ or 128), thus inflating the dot-product cost by roughly $4.5\times$ to $9\times$ compared to MHA/MQA/GQA.

TPA. Recall that TPA factorizes $\mathbf{Q}$ and $\boldsymbol{K}$ into rank-(R_q, R_k) terms (see Section 1), potentially avoiding large $\mathbf{Q}, \boldsymbol{K}$ materializations. At inference, TPA's dot-product cost can be broken into two parts:

$$\underbrace{R_q\, R_k\, d_h\, T}_{\text{QK } \mathbf{b}\text{-dot products (shared across all heads)}} + 2 \underbrace{R_q\, R_k\, h\, T}_{\text{per-head scalar multiplications}},$$

where T is the current sequence length. For concrete values $d_h = 128$, $h = 64$, $R_q = 8$, and $R_k = 2$ (or $R_q = 16$, $R_k = 1$), we obtain:

$$\text{MHA, MQA, GQA:} \quad 128 \times 64 \times T = 8192\,T,$$
$$\text{MLA:} \quad 576 \times 64 \times T = 36{,}384\,T,$$
$$\text{TPA:} \quad \big(8 \times 2 \times 128 \times T\big) + \big(2 \times 8 \times 2 \times 64 \times T\big) = 4096\,T.$$

Thus, in this setup, TPA can significantly reduce the flops needed for computing the $\mathbf{Q}(\mathbf{x}_T)\boldsymbol{K}^\top$ operation at each decoding step. The actual end-to-end wall-clock speedup also depends on kernel fusion, caching strategies, and hardware implementation details, but the factorized formulation offers a pathway to more efficient decoding than standard attention.

References

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