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):

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

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

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

Their dot-product for tokens t, τ is

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

A custom kernel could compute this sum directly, though whether it outperforms a conventional GEMM depends on the ratio $\frac{R_q}{d_h}$, $\frac{R_k}{d_h}$, hardware parallelization, etc.

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 n:

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

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

TPA: Naïve Implementation.

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

Typically $R_q, R_k, R_v \ll d_h$, so the overhead of constructing factors is small relative to $\mathcal{O}(N^2\,d_{\text{model}})$. 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:

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

and for each key:

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

Their dot product:

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

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 N queries and N keys, that is $\mathcal{O}(N^2\,R_q\,R_k\,d_h)$ 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 N^2 token pairs, this stage costs:
   $$\mathcal{O}\big(N^2\,R_qR_k\,d_h\big).$$
   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 N^2 token pairs and $R_q R_k$ rank pairs for each of the h heads, this step costs:
   $$\mathcal{O}\big(h\,N^2\,R_q R_k\big).$$

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

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

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

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

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

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

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

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

Rearrange sums:

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

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 $\mathcal{O}(N\,R_v\,d_h)$.

Stage 2: Weighted Summation by $\alpha_{t,\tau}\,a_v^{(u)}(\mathbf{x}_\tau)$. For each token t, we do

$$\sum_{\tau=1}^{N} \alpha_{t,\tau} \sum_{u=1}^{R_v} a_v^{(u)}(\mathbf{x}_\tau)\,\mathbf{b}_v^{(u)}(\mathbf{x}_\tau) \;=\; \sum_{u=1}^{R_v}\left[\sum_{\tau=1}^{N} \alpha_{t,\tau}\,a_v^{(u)}(\mathbf{x}_\tau)\right]\mathbf{b}_v^{(u)}(\mathbf{x}_\tau).$$

Naively, for each (t, u) pair, we must accumulate a sum of N vectors $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)$. Each such vector is d_h-dimensional. Thus the total cost is $\mathcal{O}(N\,R_v\,d_h)$ *per token* t, i.e. $\mathcal{O}(N^2\,R_v\,d_h)$ overall for the entire sequence. In practice, if $\mathbf{b}_v^{(u)}(\mathbf{x}_\tau)$ are precomputed for all τ, then each accumulation is a simple multiply-add loop. This is smaller than the $\mathbf{Q}\mathbf{K}^\top$ cost if $R_v \ll d_h$.

1.6 Overall Complexity for Single-Head

Combining these steps, the total cost (ignoring smaller overheads like softmax) is:

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

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

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

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

Hence, for a single head, we get:

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

If $R_q \, R_k$ and R_v are much smaller than d_h, this can be advantageous compared to explicitly forming (and then multiplying) $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, whose typical dot-product step alone is $\mathcal{O}(N^2 \, d_h)$.

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

When extending to h heads and batch size B, we multiply the sequence-related terms by $\sim B\,h$. However, crucial $\mathbf{b}$-dot products can be partially reused or shared. Specifically:

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, then each pairwise dot product $\langle \mathbf{b}_q^{(r)}(\mathbf{x}_t), \, \mathbf{b}_k^{(s)}(\mathbf{x}_\tau) \rangle$ is computed just once. That cost remains $\mathcal{O}(B \, N^2 \, R_q \, R_k \, d_h)$, not multiplied by h. After caching these dot products, each head only pays $\mathcal{O}(h \, B \, N^2 \, R_q \, R_k)$ for the scalar multiplications.

V $\mathbf{b}$-Evaluations. Likewise, since the $\mathbf{b}_v^{(u)}$ factors are shared across heads (i.e. a single set of $\mathbf{b}_v$-vectors for all heads), we only pay $\mathcal{O}(B \, n \, R_v \, d_h)$ once. Each head i then has distinct scalar factors $a_{v,i}^{(u)}(\mathbf{x}_\tau)$, leading to $\mathcal{O}(B \, N^2 \, h \, R_v \, d_h)$ for the final accumulations. Overall, the total flops can be summarized as

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

Compared to standard multi-head attention's $\mathcal{O}\big(B \, N^2 \, h \, d_h\big)$ dot-product plus $\mathcal{O}\big(B \, N^2 \, h \, d_h\big)$ multiply-by-$\mathbf{V}$, the factorization saves flops if $R_q \, R_k \ll d_h$ (for QK) and $R_v \ll d_h$ (for V). Of course, real performance depends on hardware efficiency and the overhead of these multi-stage kernels.

Discussion. By retaining Q, K, and 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, large $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ tensors (of size $n \times d_h$) 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 ensure that *(i)* the factor ranks (R_q, R_k, R_v) are sufficiently small relative to d_h, and *(ii)* the specialized kernels are efficiently implemented (i.e. we do not lose the benefits of highly optimized GEMM libraries).

In summary, **fully factorized QKV attention** can provide a path to skip forming large intermediate tensors at every step in attention. Its viability in practice depends on the ratio of rank to head dimension and on how well hardware-optimized kernels handle these "two-step" procedures for QK (and the analogous accumulation for V). When $R_q, R_k, R_v \ll d_h$, this factorized approach can lead to substantial computational savings and memory footprint reduction, complementing the KV-caching gains discussed before.

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.