Self-Attention at Constant Cost per Token via Symmetry-Aware Taylor Expansion

Franz A. Heinsen[1], Leo Kozachkov[2]

[1]GlassRoom Software LLC
[2]Brown University

January 29, 2026

Abstract

The most widely used artificial intelligence (AI) models today are Transformers employing self-attention. In its standard form, self-attention incurs costs that increase with context length, driving demand for storage, compute, and energy that is now outstripping society's ability to provide them. To help address this issue, we show that self-attention is efficiently computable to arbitrary precision with constant cost per token, achieving orders-of-magnitude reductions in memory use and computation. We derive our formulation by decomposing the conventional formulation's Taylor expansion into expressions over symmetric chains of tensor products. We exploit their symmetry to obtain feed-forward transformations that efficiently map queries and keys to coordinates in a minimal polynomial-kernel feature basis. Notably, cost is fixed inversely in proportion to head size, enabling application over a greater number of heads per token than otherwise feasible. We implement our formulation and empirically validate its correctness.[1] Our work enables unbounded token generation at modest fixed cost, substantially reducing the infrastructure and energy demands of large-scale Transformer models. The mathematical techniques we introduce are of independent interest.

1 Introduction

Most artificial intelligence (AI) services today are applications of Transformer models employing self-attention, a mathematical mechanism for capturing sequential dependencies over tokens [19, 20]. In its conventional formulation, self-attention has $\mathcal{O}(n)$ space and time complexity per token, where n is the number of tokens in the sequence, or context. The memory and compute required to process each additional token increase in proportion to the number of tokens in the sequence. The time complexity for the sequence as a whole is quadratic, $\mathcal{O}(n^2)$. All else remaining the same, processing each new token in a context requires more computing infrastructure. End-users, who benefit from longer personalized context histories and step-by-step reasoning chains, seek out AI services that can successfully handle an ever greater number of tokens. With global adoption of AI services increasing rapidly, demand for new data centers and energy sources is outstripping society's current ability to provide them [11].

Numerous modifications and replacements for conventional self-attention have been proposed to address its ever-rising memory and compute requirements. Proposed modifications include data and parameter reuse schemes, local and structured context windows, and low-rank and sparse approximations [18, 21,

[1]Source code and replication instructions are at https://github.com/glassroom/sata_attention.

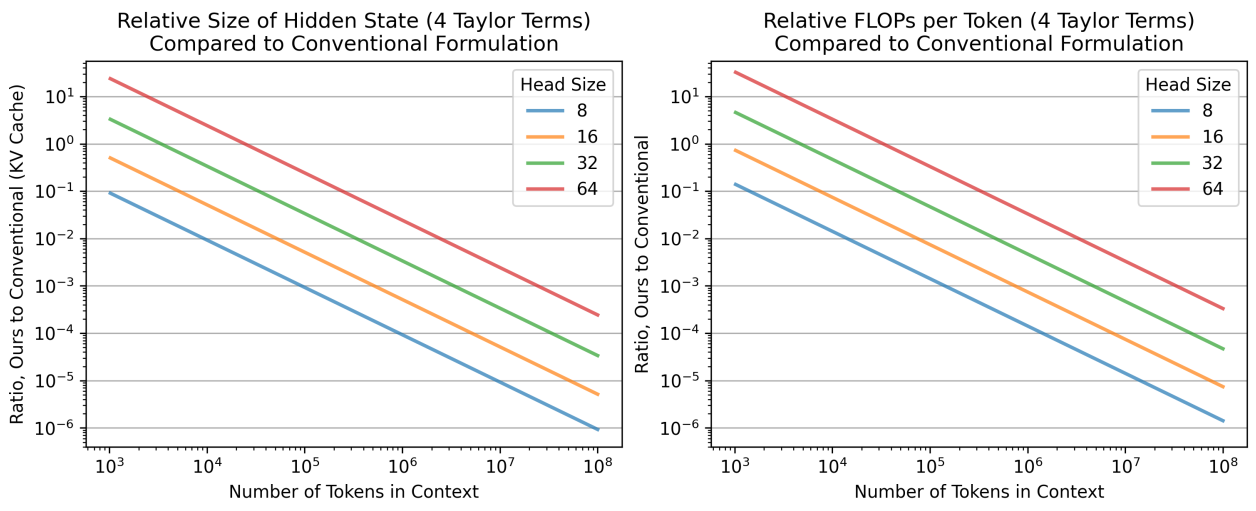


Figure 1: Hidden state size and FLOPs per token (4 Taylor terms), compared to conventional formulation.

22]. The proposed replacements are mainly recurrent neural networks (RNNs) with linear recurrences computable via parallel scan [2, 3], including state space models (SSMs) [1, 5–7, 9].

Rather than propose another modification or replacement, we show that self-attention can be efficiently computed at any desired precision at constant cost per token, $\mathcal{O}(1)$, with a hidden state of size

$$(d_V + 1)\binom{d_K + P - 1}{P - 1}, \quad \text{// fixed number of elements in hidden state} \tag{1}$$

and a constant number of floating-point operations (FLOPs) per token in the forward pass,

$$\left(4d_V + \frac{2\,(P\,d_K + 1)}{d_K + 1} + 2\right)\binom{d_K + P - 1}{P - 1}, \quad \text{// fixed FLOPs per token in forward pass} \tag{2}$$

where d_K and d_V are key and value sizes, respectively, and P is the number of terms in a Taylor series expansion, controlling precision. In practice, we find that four Taylor terms ($P = 4$) suffice for recovering conventional attention with elementwise errors of approximately the same magnitude as Float16 resolution, acceptable for many AI applications. As we increase the number of tokens in context, our formulation's costs per token become orders of magnitude more efficient than previously possible (Figure 1). Our formulation is a form of linear attention [9], computable via parallel scan.

In comparison, the conventional formulation has a hidden state of variable size, the key–value cache ("KV cache," for short), with $n(d_K + d_V)$ elements [14], where n denotes the number of tokens in context, executes a variable number of FLOPs per token in the forward pass, $n(2d_K + 2d_V + 3)$ [8], and cannot be computed via parallel scan, because its stepwise transformation is not associative over tokens.

Our formulation enables Transformer models to generate tokens in perpetuity with modest fixed costs per token, at any desired precision, so long as the Transformer's positional encoding scheme can accommodate sequences of indefinite length. New levers for trading off memory use, compute cost, and numerical precision become available. The number of Taylor terms, P, controls precision. Space and time complexity are inversely proportional to d_K and d_V, such that we can reduce the fixed per-token costs by making heads smaller and applying self-attention over a greater number of heads. Alternatively, if given a fixed budget, we can apply self-attention over a greater number of heads per token than otherwise feasible. In contrast, with the conventional formulation, space and time complexity are proportional to the number of heads, because each head's per-token costs are proportional to context length.

Previous efforts to approximate self-attention via Taylor expansion have stopped at the quadratic term (*i.e.*, second order) due to the perceived complexity of evaluating all necessary polynomial interactions for higher-degree terms [1, 4, 12, 15, 23]. We show that the Taylor expansion decomposes into symmetric chains of tensor products, and their symmetric structure naturally reveals the minimal basis for all polynomial interactions. Our key contribution is a maximally succinct, computationally efficient, and embarrassingly parallel feed-forward transformation that evaluates the associated kernel function to arbitrary Taylor truncation order at constant cost per token.

We implement our formulation, verify that it recovers attention with increasing accuracy as we expand the Taylor series, and confirm that it reduces memory use and run time by orders of magnitude as we increase context length, compared to the conventional formulation. Our results likely understate the achievable savings in optimized implementations. All evidence indicates that our work here can substantially reduce the infrastructure and energy demands of large-scale Transformer models.

2 Deriving Our Formulation

2.1 Taylor Expansion of Core Operation

The core operation in self-attention consists of the dot-product of a query and a key vector, scaling the result, and applying the exponential function to it. Given query and key vectors $q, k \in \mathbb{R}^{d_K}$ and a conventional constant $c = \sqrt{d_K}$, the core operation's Taylor expansion is:

$$\exp\left(\frac{q^\top k}{c}\right) = \sum_{p=0}^{\infty} \alpha_p \, (q^\top k)^p, \qquad \alpha_p := \frac{1}{p! \, c^p}. \tag{3}$$

2.2 Decomposing into Expressions over Symmetric Tensors

For each integer $p > 0$ in (3), we can express $\left(q^\top k\right)^p$ as:

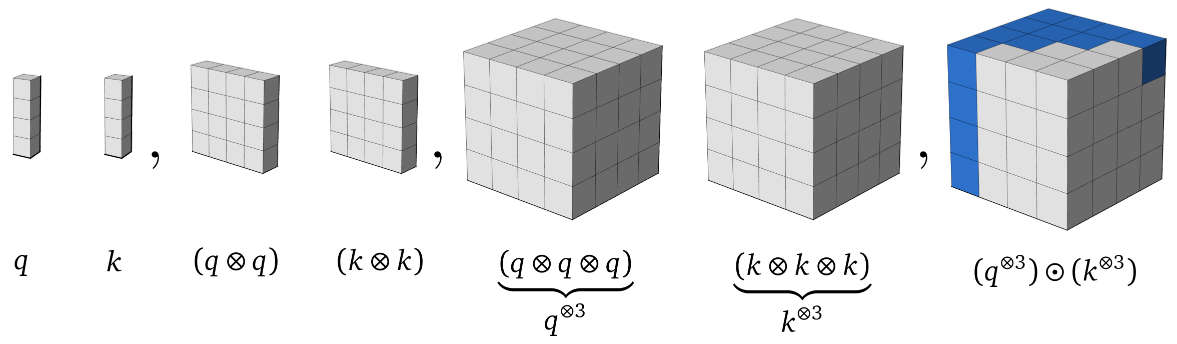

q
k
$(q \otimes q)$
$(k \otimes k)$
$\underbrace{(q \otimes q \otimes q)}_{q^{\otimes 3}}$
$\underbrace{(k \otimes k \otimes k)}_{k^{\otimes 3}}$
$(q^{\otimes 3}) \odot (k^{\otimes 3})$

Figure 2: Illustration of the symmetric tensors of increasing order in (4). The blue region in the rightmost plot contains the minimal monomial basis for representing $\left(q^\top k\right)^p$, as discussed in Section 2.3.

$$\begin{aligned}
\left(q^\top k\right)^1 &= q^\top k && = \sum_{i=1}^{d_K} q_i k_i && = \sum \underbrace{q \odot k}_{\in \mathbb{R}^{d_K}} \\
(q^\top k)^2 &= \sum_{i_1=1}^{d_K} q_{i_1} k_{i_1} \sum_{i_2=1}^{d_K} q_{i_2} k_{i_2} && = \sum_{i_1=1}^{d_K} \sum_{i_2=1}^{d_K} (q_{i_1} q_{i_2})(k_{i_1} k_{i_2}) && = \sum \underbrace{(q \otimes q) \odot (k \otimes k)}_{\in \mathbb{R}^{d_K \times d_K}} \\
(q^\top k)^3 &= \sum_{i_1=1}^{d_K} q_{i_1} k_{i_1} \sum_{i_2=1}^{d_K} q_{i_2} k_{i_2} \sum_{i_3=1}^{d_K} q_{i_3} k_{i_3} && = \sum_{i_1=1}^{d_K} \sum_{i_2=1}^{d_K} \sum_{i_3=1}^{d_K} (q_{i_1} q_{i_2} q_{i_3})(k_{i_1} k_{i_2} k_{i_3}) && = \sum \underbrace{(q \otimes q \otimes q) \odot (k \otimes k \otimes k)}_{\in \mathbb{R}^{d_K \times d_K \times d_K}}, \\
\vdots \quad & \quad \vdots && \quad \vdots && \quad \vdots
\end{aligned} \tag{4}$$

where $\odot$ denotes elementwise (Hadamard) product, and $\otimes$ denotes tensor (outer) product. See Figure 2 for a helpful illustration. Generalizing, we obtain the following expression for $p > 0$:

$$\left(q^\top k\right)^p \quad = \sum_{i_1=1}^{d_K} \sum_{i_2=1}^{d_K} \cdots \sum_{i_p=1}^{d_K} (q_{i_1} q_{i_2} \dots q_{i_p})(k_{i_1} k_{i_2} \dots k_{i_p}) \quad = \sum (q^{\otimes p}) \odot (k^{\otimes p}), \tag{5}$$

where $q^{\otimes p}$ and $k^{\otimes p}$ are order-p symmetric tensors obtained by chains of tensor products:

$$\begin{aligned}
q^{\otimes p} &= \underbrace{q \otimes q \otimes \cdots \otimes q}_{p \text{ identical factors}} \\
k^{\otimes p} &= \underbrace{k \otimes k \otimes \cdots \otimes k}_{p \text{ identical factors}}.
\end{aligned} \tag{6}$$

By construction, the elements of $q^{\otimes p}$ and $k^{\otimes p}$ are monomials multiplying all possible combinations of p scalar elements of k and q, respectively. The number of all such possible combinations is d_K^p.

2.3 Identifying the Minimal Basis in Each Symmetric Tensor

The elementwise multiplication of the two symmetric tensors, $(q^{\otimes p}) \odot (k^{\otimes p})$, in (5), evaluates to a third symmetric tensor, $(q \odot k)^{\otimes p}$, whose d_K^p elements are the monomials that sum to $\left(q^\top k\right)^p$ (Figure 2). Each of these three order-p tensors being symmetric, its upper hyper-triangular region, consisting of

$$m_p = \binom{d_K + p - 1}{p} \tag{7}$$

elements indexed by $i_1 \leq i_2 \leq \cdots \leq i_p$, contains the tensor's unique elements [10, 16]. All monomials outside that region are permutations of a monomial in the region. Therefore, the upper hyper-triangular regions of $q^{\otimes p}$ and $k^{\otimes p}$ contain the m_p unique monomials combining elements of q and k, respectively, that make up the *minimal basis* in the space of such monomials, for obtaining $\left(q^\top k\right)^p$.

To obtain the final correct expression for $\left(q^\top k\right)^p$, the elements of this minimal basis must be weighted appropriately before being summed. A simple example with $d_K = p = 2$ helps to illustrate this point. Letting $q = (q_1, q_2)$ and $k = (k_1, k_2)$:

$$(q^\top k)^2 = (q_1 k_1 + q_2 k_2)^2 = q_1^2 k_1^2 + q_1 q_2 k_1 k_2 + q_2 q_1 k_2 k_1 + q_2^2 k_2^2.$$

The middle two terms are identical, and can therefore be combined into a single term $2 q_1 q_2 k_1 k_2$. In general, we can distinguish the monomial *basis components* from the monomial *coefficients*. In the example above, $q_1 q_2 k_1 k_2$ is a basis component and 2 is its coefficient.

In summary, the expansion of $(q^\top k)^p$ can be written as a sum over the m_p distinct monomials corresponding to the upper hyper-triangular region of $(q \odot k)^{\otimes p}$, where each monomial appears with a multiplicity determined by the number of index permutations that produce it. This separation between a minimal set of basis monomials and their associated combinatorial coefficients will allow us to reformulate $(q^\top k)^p$ as a structured inner product.

2.4 From Symmetric Tensors to a Weighted Feature Inner Product

It follows from the previous subsection that the expansion of $(q^\top k)^p$ can be organized by grouping identical monomials and weighting each by its multiplicity. This naturally suggests defining a feature map $\Phi_p(\cdot)$ whose components correspond to the m_p basis monomials, together with a diagonal weighting that accounts for their combinatorial coefficients. With these definitions, $(q^\top k)^p$ can be written as a weighted inner product,

$$\left(q^\top k\right)^p = \left\langle \Phi_p(q), \Phi_p(k) \right\rangle_{C_p}, \tag{8}$$

where $\Phi_p(q)$ and $\Phi_p(k)$ denote application of a feature map $\Phi_p : \mathbb{R}^{d_K} \to \mathbb{R}^{m_p}$ that obtains the m_p monomials in the upper hyper-triangular region of $q^{\otimes p}$ and $k^{\otimes p}$, respectively, *tightly packed in a vector*; and $C_p \in \mathbb{R}^{m_p \times m_p}$ is a constant diagonal weight matrix, each diagonal element of which is the number of permutations in the full symmetric tensor corresponding to each unique monomial in the tightly packed vector, scaling that unique monomial by its frequency in the full symmetric tensor.

A well-known result from kernel theory [17] is that for each integer $p \geq 0$, there exists a reproducing kernel Hilbert space (RKHS) $\mathscr{H}_p$ and an associated feature map, such that degree-p monomials can be expressed as an inner product in $\mathscr{H}_p$. So far, we have ignored $p = 0$. We incorporate it by treating $q^{\otimes 0}$ and $k^{\otimes 0}$ as empty chains of tensor products that evaluate to scalar value 1, as is conventional, and obtain a conventional RKHS feature map, with $\Phi_0(q) = \Phi_0(k) = 1$. See Appendix A1 for the details.

Truncating for Approximation With this feature map perspective, the Taylor series (3) becomes an infinite sum of scaled weighted inner products. In practice, we truncate the series to a finite number of terms P, obtaining an approximation:

$$\exp\left(\frac{q^\top k}{c}\right) \approx \sum_{p=0}^{P-1} \alpha_p \left\langle \Phi_p(q), \Phi_p(k) \right\rangle_{C_p}. \tag{9}$$

Efficient Computation For each power p, the weight matrix, C_p, is *constant*, so the elements in its diagonal can be precomputed in advance, only once, for subsequent efficient weighting of the inner product via m_p elementwise multiplications. The indices that select an order-p upper hyper-triangular region are also *constant*, and therefore can be similarly precomputed in advance, only once, for subsequent efficient computation of $\Phi_p(\cdot)$ via parallel multiplication of elements selected from its input vector:

$$\Phi_p(x) := \left[\prod_{j=1}^{p} x_{M_{p\,ij}} \quad {}_{i=1,2,\ldots,m_p} \right], \tag{10}$$

where $M_p \in \{1, 2, \ldots, d_K\}^{m_p \times p}$ is the constant matrix with precomputed indices. In pseudo-code:

```
def Phi(x):  return x[..., M].prod(dim=-1)
```
, (11)

a maximally succinct, computationally efficient, embarrassingly parallel transformation. The matrix M_p, each row containing indices $i_1, i_2, \ldots, i_p$, has a hierarchical structure, given by $i_1 \le i_2 \le \cdots \le i_p$, opening additional opportunities for improving computational efficiency, which we leave for future work.[2]

By grouping identical monomials, tightly packing them into vectors, and weighting each unique monomial by its multiplicity, our method reduces the number of features we must store and the number of floating-point operations (FLOPs) we must execute, per token, *by orders of magnitude*, compared to a feature map that naively evaluates all possible permutations of the unique monomials (Figure 3).

2.5 Linear Attention with Our Formulation

So far, we have described how the exponentiated dot product can be approximated as a weighted sum of inner products. Compared to conventional Linear Transformer formulations (e.g., [9]), the key distinction is that our approximation consists of multiple inner products (one per term in the truncated Taylor expansion) rather than a single inner product. As a result, whereas standard Linear Transformers can be evaluated using a single parallel prefix scan, our approach requires P such scans, corresponding to

[2]It's not too hard to show that for $p > 0$, every monomial indexed by M_{p-1} is a factor of a monomial indexed by M_p.

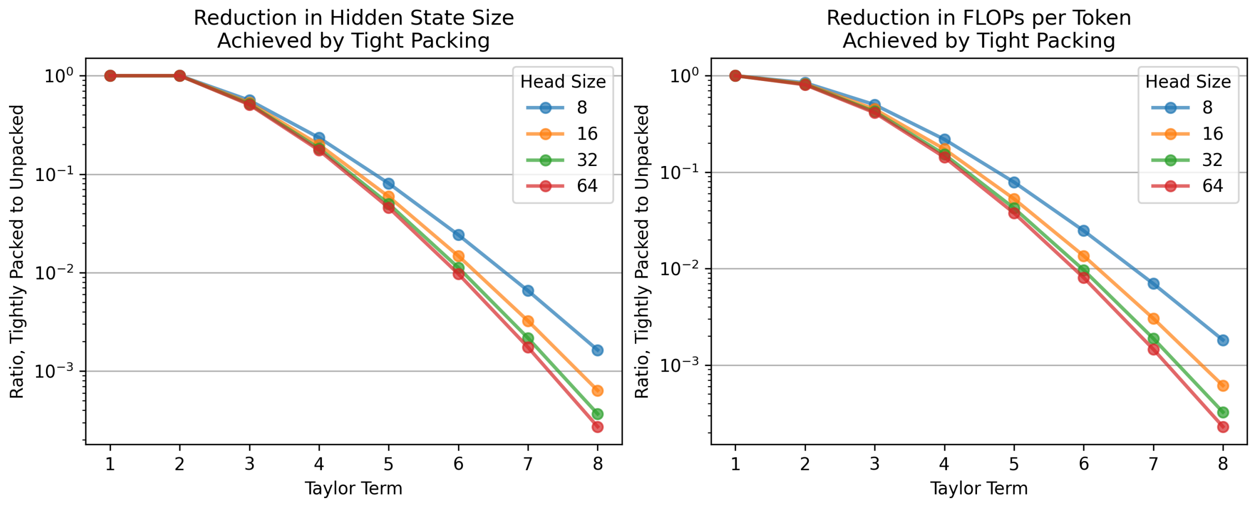


Figure 3: Reduction in hidden state size and FLOPs per token achieved by our tight packing method.

the P Taylor terms. Importantly however, these scans are independent of each other and can be executed in parallel, after which their outputs are aggregated to form the final approximation.

Please see Appendix A2 for the details.

2.6 Hidden State Size

We define "hidden state size" as the number of elements in accumulated state, per token, for linear attention (Appendix A2). For one Taylor term with degree p, hidden state size is:

$$(d_V + 1)\binom{d_K + p - 1}{p}. \quad \text{// hidden state size, one Taylor term} \tag{12}$$

For all Taylor terms in a model, hidden state size is $\sum_{p=0}^{P-1}\left((d_V+1)\binom{d_K+p-1}{p}\right)$, equal to:

$$(d_V + 1)\binom{d_K + P - 1}{P - 1}. \quad \text{// hidden state size, all Taylor terms} \tag{13}$$

Compared to a naive feature map that computes d_K^p monomials for a Taylor term of degree p, our formulation reduces hidden state size by orders of magnitude (Figure 3, left panel). The conventional formulation of attention's equivalent to a hidden state is the KV cache, consisting of $n(d_K + d_V)$ elements [14], where n denotes the number of tokens in context. The left panel of Figure 1 compares our formulation's hidden state size (for 4 Taylor terms) to the size of the conventional formulation's KV cache, for different head sizes and context lengths.

2.7 FLOPs per Token

We define floating-point operations (FLOPS) per token as the number of multiply and add operations that must be executed in the forward pass to compute attention for the token. Adding up the number of those operations, FLOPs per token for one Taylor term with degree p is:

$$(4d_V + 2p + 4)\binom{d_K + p - 1}{p}. \quad \text{// FLOPs per token, one Taylor term} \tag{14}$$

For all Taylor terms in a model, FLOPs per token is $\sum_{p=0}^{P-1}\left((4d_V + 2p + 4)\binom{d_K + p - 1}{p}\right)$, equal to:

$$\left(4d_V + \frac{2\,(P\,d_K + 1)}{d_K + 1} + 2\right)\binom{d_K + P - 1}{P - 1}. \quad \text{// fixed FLOPs per token, all Taylor terms} \tag{15}$$

Compared to a naive feature map that computes d_K^p monomials for a Taylor term of degree p, our formulation reduces FLOPs per token by orders of magnitude (Figure 3, right panel). The conventional formulation of attention executes $n(2d_K + 2d_V + 3)$ FLOPS per token in the forward pass, where n denotes the number of tokens in context.[3]. The right panel of Figure 1 compares our formulation's FLOPs per token in a forward pass (for 4 Taylor terms) to that of the conventional formulation, for different head sizes and context lengths.

2.8 Costs Fixed Inversely in Proportion to Head Size

Our formulation fixes costs per token inversely in proportion to d_K and d_V, enabling us to reduce costs by making heads smaller and applying attention over a greater number of heads than otherwise feasible, given an embedding size (Figure 4). In contrast, with the conventional formulation, memory use and run time per token increase in proportion to the number of heads, because each head's per-token costs are proportional to context length, limiting the number of heads that are feasible in practice.

3 Implementation

We implement our formulation, and verify that it successfully recovers self-attention computed conventionally. We also verify that our implementation successfully reduces memory use and run time by orders of magnitude, compared to those of the conventional formulation. Our implementation relies on PyTorch, a commonly used software framework for parallel computation [13].

3.1 Successful Recovery of Conventional Self-Attention

We measure our implementation's ability to recover conventional self-attention over sequences with up to 100K tokens, and successfully verify that each additional Taylor term we add to the series decreases reconstruction error. See Appendices A3 and A4 for detailed reconstruction results.

We find that four Taylor terms, $P = 4$, suffice for recovering conventional attention with elementwise errors of roughly the same magnitude as Float16 resolution, which is not too surprising, because the magnitude of scaling constant α_p declines to that resolution by the fourth Taylor Term (Figure 5).

[3]We measure forward-pass FLOPs per token for the conventional formulation with the same method as [8]: (a) FLOPs for computing the QK^T logits, equal to $2 \times n_{\text{queries}} \times n_{\text{tokens}} \times d_K \times n_{\text{heads}}$; plus (b) FLOPs for applying a Softmax function over each row of scores, equal to $3 \times n_{\text{queries}} \times n_{\text{tokens}}$; plus (c) FLOPs for contracting the attention matrix with values, equal to $2 \times n_{\text{queries}} \times n_{\text{tokens}} \times d_V \times n_{\text{heads}}$. We specify $n_{\text{queries}} = 1$ and $n_{\text{heads}} = 1$, and simplify the sum of (a), (b), and (c).

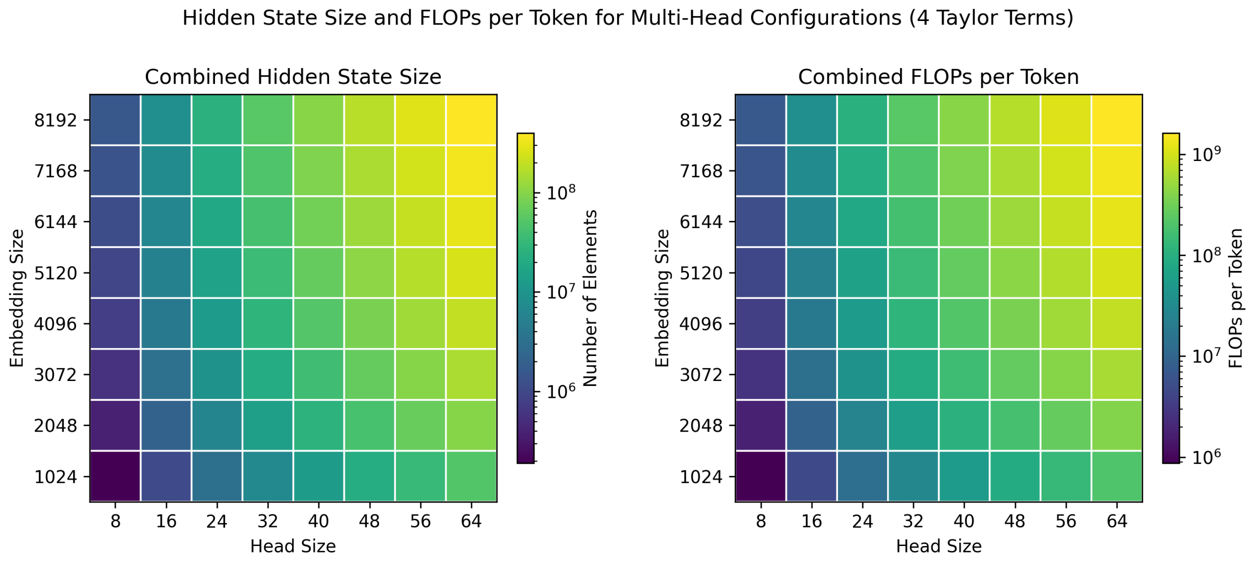


Figure 4: Size of hidden state and FLOPs per token in multi-head configurations.

3.2 Successful Reduction of Memory Use and Run Time by Orders of Magnitude

We measure peak memory allocated and run time, per token, of our proof-of-concept implementation, compared to a conventional implementation, given a context that we increase in length from 1K to 100M tokens, with four different head sizes, on an Nvidia GPU. All operations are executed with autocasting, *i.e.*, letting PyTorch decide which floating-point format to use, as necessary, for matching the precision required by each operation. The peak memory allocation figures include temporary space consumed by interim data. Run time is measured as the mean of seven executions.

Both memory use and run time of our implementation are unfairly penalized by the issues discussed in 3.3. Despite these unfair disadvantages, our implementation's peak memory allocated, per token, declines three orders of magnitude below that of conventional attention as we increase context length to 100M tokens (Figure 6, left panel). Run time, per token, declines almost three orders of magnitude below conventional attention as we increase context length to 100M tokens (Figure 6, right panel).

3.3 Proof of Concept

Our implementation is an initial one, and should properly be considered a proof of concept. Its source code consists primarily of high-level calls to PyTorch's preexisting libraries, without any customization. Unlike implementations of the conventional formulation, which has now benefited from nearly a decade of performance optimization work by a large community of AI researchers and practitioners, our formulation is new and has yet to benefit from comparable long-term work.

Targets for performance optimization include:

Unnecessary Temporary Copying of Query and Key Elements *Currently, we make $m_p \times p$ temporary copies of elements from each query and key vector, instead of pointing to those elements,* for mapping the vector to m_p monomial features. We rely on PyTorch's advanced indexing facilities to obtain views of

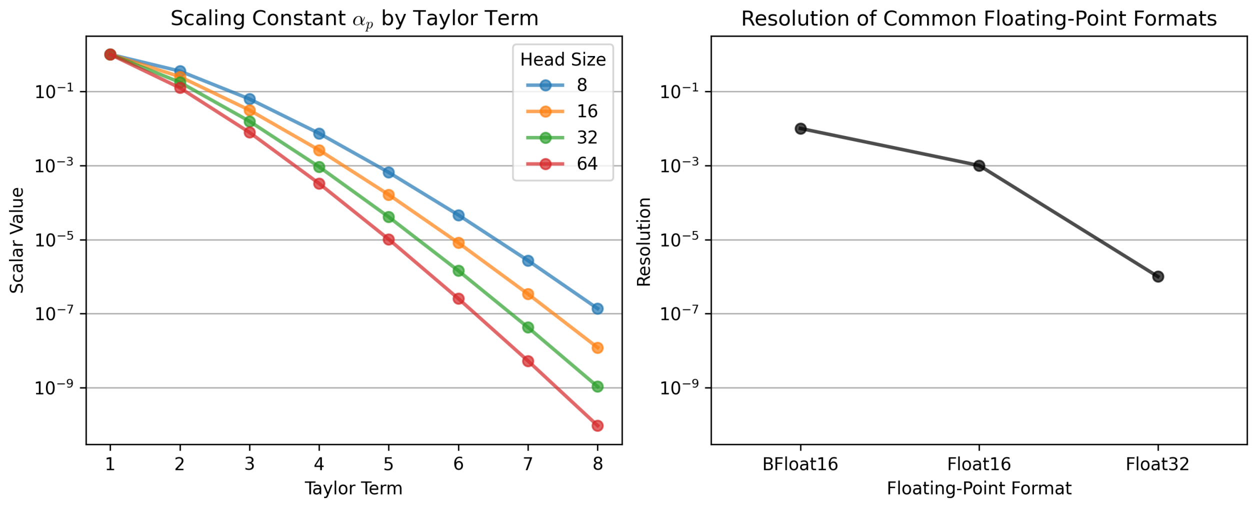


Figure 5: Scaling constant of each Taylor term, α_p, compared to resolution of common floating-point formats.

shape $m_p \times p$ from every vector, but each "view" returned by PyTorch actually makes a copy of the data, which can saturate memory bandwidth. In principle, copying all that data is unnecessary.

Absence of Optimizations that Exploit Hierarchical Structure of Symmetric Indices *Currently, we do not exploit the structure of the indices stored in matrix M_p.* Each row of M_p contains indices $i_1, i_2, \ldots, i_p$ organized hierarchically by $i_1 \leq i_2 \leq \cdots \leq i_p$, opening additional opportunities for improving computational efficiency. In principle, it should be possible to exploit hierarchical structure to reduce memory and compute use.

Sequential Instead of Parallel Evaluation of Taylor Terms *Currently, we evaluate Taylor Terms sequentially, instead of in parallel.* We queue them on a single stream on an Nvidia GPU. As we discuss in 2.5, Taylor terms can be evaluated in parallel, because they are independent of each other.

Absence of Common On-Device Optimizations *Currently, our focus is on validating the correctness of our formulation, not on developing a high-performance implementation.* A more efficient implementation requires writing low-level on-device code (*e.g.*, a CUDA kernel for Nvidia GPUs) that carefully handles data, both to avoid unnecessarily making temporary copies of it, and to ensure it is more frequently on faster-access memory (*e.g.*, HBM instead of SRAM on Nvidia GPUs) as needed for computation.

Absence of Additional Performance Optimizations *Currently, we have not explored any additional performance optimizations.* They include the possibility of reducing the dimensionality of higher-order feature spaces (say, for $P \geq 4$) by applying conventional techniques, such as dropping basis components with low-magnitude coefficients in C_p, finding best-fit lower-rank basis approximations, and obtaining fast approximations of the basis via random sampling or random projections.[4]

[4]We should add that many methods for improving the conventional formulation's space and time complexity also benefit our formulation (*e.g.*, data and parameter reuse schemes like sharing a single key-value head with multiple query heads).

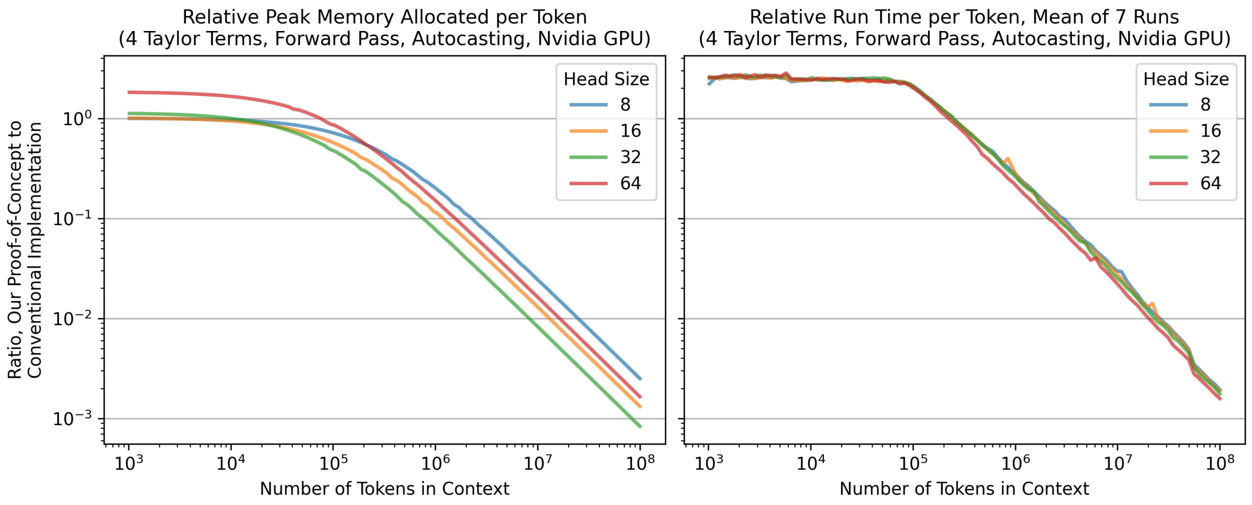


Figure 6: Our proof-of-concept implementation's memory use and run time per token, compared to those of a conventional formulation. Our implementation is unfairly penalized by the issues discussed in 3.3.

4 Discussion

We have presented a formulation of self-attention that approximates the exponential kernel underlying its Softmax function while eliminating its dependence on context length. For a fixed approximation order, both memory use and computation become constant per token. Rather than modifying the definition of attention, we reorganize its computation by approximating every exponentiated dot product with a truncated Taylor expansion and expressing the resulting terms in a compact, symmetry-aware feature representation. As the approximation order increases to infinity, our construction converges to conventional attention, allowing it to be recovered to arbitrary accuracy subject to numerical precision.

Implications for long-context inference The most immediate consequence of approximating the exponential kernel with our method is that the marginal cost of extending context no longer increases with sequence length. In conventional attention, longer contexts require proportionally larger key–value caches and proportionally more computation per generated token, increasing pressure on memory capacity, processor utilization, communications bandwidth, and energy consumption. Our formulation replaces the variable-size cache with a fixed-size accumulated state whose size and cost depend only on head dimensions and the chosen approximation order.

A second implication is architectural. Because per-token state size and computation decrease in proportion to head dimension, attention can be distributed across many smaller heads without incurring the linear growth in cost that arises in conventional multi-head attention. While we do not evaluate trained models here, the formulation makes such configurations feasible in regimes where key–value caching would otherwise dominate resource usage.

Accuracy–efficiency trade-offs Our approach introduces an explicit accuracy–efficiency trade-off through the choice of approximation order. Unlike sparsity-based or low-rank methods, which alter the attention pattern itself, our method targets the same attention pattern and approximates it directly.

In practice, we find that only a small number of terms is sufficient to match conventional attention to within typical reduced-precision numerical error. This suggests that, for many inference settings, higher-order contributions to the kernel fall below the noise floor imposed by floating-point arithmetic. Nevertheless, the appropriate approximation order may depend on model architecture, head size, and representation statistics, and should be treated as a tunable hyperparameter.

Relation to prior work The proposed method fits naturally within the family of linear attention mechanisms, in which attention can be computed via streaming accumulation rather than explicit storage of past keys and values. The main distinction is that we approximate the exponential kernel as a sum of multiple, independently accumulated components, rather than attempting to represent it with a single feature map. This perspective clarifies why previous Taylor-based approaches typically stopped at very low order: without exploiting symmetry, the number of polynomial features grows combinatorially. By identifying and tightly packing the minimal set of unique monomials, we make higher-order approximations practical at modest head sizes.

Limitations and Future Directions Our implementation is intentionally a proof of concept and leaves several questions open. First, our benchmarks isolate the attention mechanism itself and do not measure end-to-end throughput in trained models, where other components may dominate runtime. Second, our recovery experiments verify correctness using synthetic inputs, but do not evaluate downstream task performance or training dynamics.

In addition, the current implementation incurs overheads that are not inherent to the method, including temporary data copies and sequential evaluation of approximation terms. These factors penalize performance relative to what a fused, hardware-optimized implementation could achieve, and the reported runtime results should therefore be interpreted conservatively.

Although the attention mechanism itself incurs constant per-token cost for a fixed approximation order, unbounded-context generation also requires positional encoding schemes that remain well-behaved at extreme sequence lengths. Such schemes are complementary to, rather than replaced by, the proposed formulation.

Finally, head dimension plays a more prominent role than in conventional attention. The formulation is most favorable for small heads and modest approximation orders, and should be understood as opening a new region of the architectural design space rather than uniformly improving all configurations.

Several directions for future work follow naturally. An important next step is to train Transformer models, end-to-end and via transfer learning, using the proposed attention mechanism, and evaluate downstream performance and convergence behavior. Doing so at large scale will require developing fused hardware kernels that target the performance optimizations we discuss in 3.3. Further opportunities include compressing higher-order feature spaces and extending the approach to other analytic kernels beyond the exponential. More generally, our work enables exploration of new approaches to address the rising demand for storage, compute, and energy, driven by growing adoption of AI services.

References

[1] S. Arora, S. Eyuboglu, M. Zhang, A. Timalsina, S. Alberti, D. Zinsley, J. Zou, A. Rudra, and C. Ré. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint*

arXiv:2402.18668, 2024.

[2] G. E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, Nov. 1990.

[3] G. E. Blelloch. *Vector Models for Data-Parallel Computing*. MIT Press, Cambridge, MA, 1990. ISBN 026202313X.

[4] J. Dass, S. Wu, H. Shi, C. Li, Z. Ye, Z. Wang, and Y. Lin. Vitality: Unifying low-rank and sparse approximation for vision transformer acceleration with a linear taylor attention. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 415–428. IEEE, 2023.

[5] L. Feng, F. Tung, M. O. Ahmed, Y. Bengio, and H. Hajimirsadeghi. Were RNNs all we needed?, 2024.

[6] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

[7] A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces, 2022.

[8] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. Training compute-optimal large language models, 2022.

[9] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are RNNs: Fast autoregressive Transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[10] A. Kostrikin, I. Manin, and Y. Manin. *Linear Algebra and Geometry*. Algebra, logic, and applications. Taylor & Francis, 1989. ISBN 9782881246838.

[11] N. Maslej, L. Fattorini, R. Perrault, Y. Gil, V. Parli, N. Kariuki, E. Capstick, A. Reuel, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, J. C. Niebles, Y. Shoham, R. Wald, T. Walsh, A. Hamrah, L. Santarlasci, J. B. Lotufo, A. Rome, A. Shi, and S. Oak. The AI Index 2025 Annual Report. Technical report, AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, 2025.

[12] T. C. Nauen, S. Palacio, and A. Dengel. Taylorshift: Shifting the complexity of self-attention from squared to linear (and back) using taylor-softmax. In *International Conference on Pattern Recognition*, pages 1–16. Springer, 2025.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library, 2019.

[14] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean. Efficiently scaling Transformer inference, 2022.

[15] Y. Qiu, J. Yan, W. Zhang, J. Zhou, and J. Zhang. Mb-taylorformer: Multi-branch efficient transformer expanded by taylor formula for image dehazing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12802–12812, 2023.

[16] M. D. Schatz, T. M. Low, R. A. van de Geijn, and T. G. Kolda. Exploiting symmetry in tensors for high performance: Multiplication with symmetric tensors. *SIAM Journal on Scientific Computing*, 36(5):C453–C479, 2014. doi: 10.1137/130907215.

[17] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[18] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient Transformers: a survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

[19] R. E. Turner. An introduction to Transformers, 2024.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

[21] L. Weng. The Transformer Family (survey of transformer architectures), 2020. URL https://lilianweng.github.io/posts/2020-04-07-the-transformer-family/. Accessed on January 19, 2026.

[22] L. Weng. The Transformer Family Version 2.0 (survey of transformer architectures), 2023. URL https://lilianweng.github.io/posts/2023-01-27-the-transformer-family-v2/. Accessed on January 19, 2026.

[23] Y. Xu, C. Li, D. Li, X. Sheng, F. Jiang, L. Tian, and E. Barsoum. Qt-vit: Improving linear attention in vit with quadratic taylor expansion. *Advances in Neural Information Processing Systems*, 37: 83048–83067, 2024.

Appendix

A1 Incorporating Degree Zero in Feature Map

We incorporate the first Taylor term, with $p = 0$, by treating $q^{\otimes 0}$ and $k^{\otimes 0}$ as empty chains of tensor products that evaluate to scalar value 1, as is conventional, obtaining:

$$\begin{aligned} m_0 &= \binom{d_K + 0 - 1}{0} && = 1 \\ M_0 &= m_0 \times 0 \text{ matrix} && = [\,] \\ \Phi_0(\cdot) &= (\text{empty product}) && = 1. \end{aligned} \tag{16}$$

With this change, Φ_p becomes a conventional RKHS feature map from vectors in $\mathbb{R}^{d_K}$ to m_p coordinates in the minimal basis of monomials for obtaining $\left(q^\top k\right)^p$, for all $p \geq 0$.

A2 Linear Attention

Consider causal self-attention over a sequence

$$\{(q_t, k_t, v_t)\}_{t=1}^T, \qquad q_t, k_t \in \mathbb{R}^{d_K}, \quad v_t \in \mathbb{R}^{d_V}.$$

The causal attention output at time T is

$$y_T = \frac{\sum_{t=1}^{T} \exp\left(\frac{q_T^\top k_t}{c}\right) v_t}{\sum_{t=1}^{T} \exp\left(\frac{q_T^\top k_t}{c}\right)}.$$

We approximate the exponential kernel using the truncated expansion (9). Substituting this approximation into the attention expression yields

$$y_T \approx \frac{\sum_{t=1}^{T} \sum_{p=0}^{P-1} \alpha_p \left\langle \Phi_p(q_T), \Phi_p(k_t) \right\rangle_{C_p} v_t}{\sum_{t=1}^{T} \sum_{p=0}^{P-1} \alpha_p \left\langle \Phi_p(q_T), \Phi_p(k_t) \right\rangle_{C_p}}.$$

Reordering the sums gives

$$y_T \approx \frac{\sum_{p=0}^{P-1} \alpha_p \sum_{t=1}^{T} \left\langle \Phi_p(q_T), \Phi_p(k_t) \right\rangle_{C_p} v_t}{\sum_{p=0}^{P-1} \alpha_p \sum_{t=1}^{T} \left\langle \Phi_p(q_T), \Phi_p(k_t) \right\rangle_{C_p}}. \tag{17}$$

Accumulated Feature States The inner product is linear in its second argument, so the denominator terms factor as

$$\sum_{t=1}^{T} \left\langle \Phi_p(q_T), \Phi_p(k_t) \right\rangle_{C_p} = \left\langle \Phi_p(q_T), \sum_{t=1}^{T} \Phi_p(k_t) \right\rangle_{C_p} .$$

The numerator is vector-valued due to the presence of v_t, motivating the introduction of degree-wise accumulated states. For each degree p, define

$$Z_{p,T} := \alpha_p \sum_{t=1}^{T} \Phi_p(k_t), \qquad S_{p,T} := \alpha_p \sum_{t=1}^{T} \Phi_p(k_t)\, v_t^{\top}.$$

These states satisfy the linear recurrences

$$Z_{p,T} = Z_{p,T-1} + \alpha_p \Phi_p(k_T), \qquad S_{p,T} = S_{p,T-1} + \alpha_p \Phi_p(k_T) v_T^{\top}.$$

Both updates are linear and can therefore be computed efficiently via either sequential recurrence or parallel prefix-sum (scan).

Evaluation at the Query By construction,

$$S_{p,T}^{\top} h = \alpha_p \sum_{t=1}^{T} \left\langle h, \Phi_p(k_t) \right\rangle_{C_p} v_t \qquad \text{for any } h.$$

Evaluating at $h = \Phi_p(q_T)$ gives

$$S_{p,T}^{\top} \Phi_p(q_T) = \alpha_p \sum_{t=1}^{T} \left\langle \Phi_p(q_T), \Phi_p(k_t) \right\rangle_{C_p} v_t,$$

which matches the numerator contributions in (17). Similarly,

$$\left\langle \Phi_p(q_T), Z_{p,T} \right\rangle_{C_p} = \alpha_p \sum_{t=1}^{T} \left\langle \Phi_p(q_T), \Phi_p(k_t) \right\rangle_{C_p} .$$

Final Attention Expression Aggregating over all degrees p, define

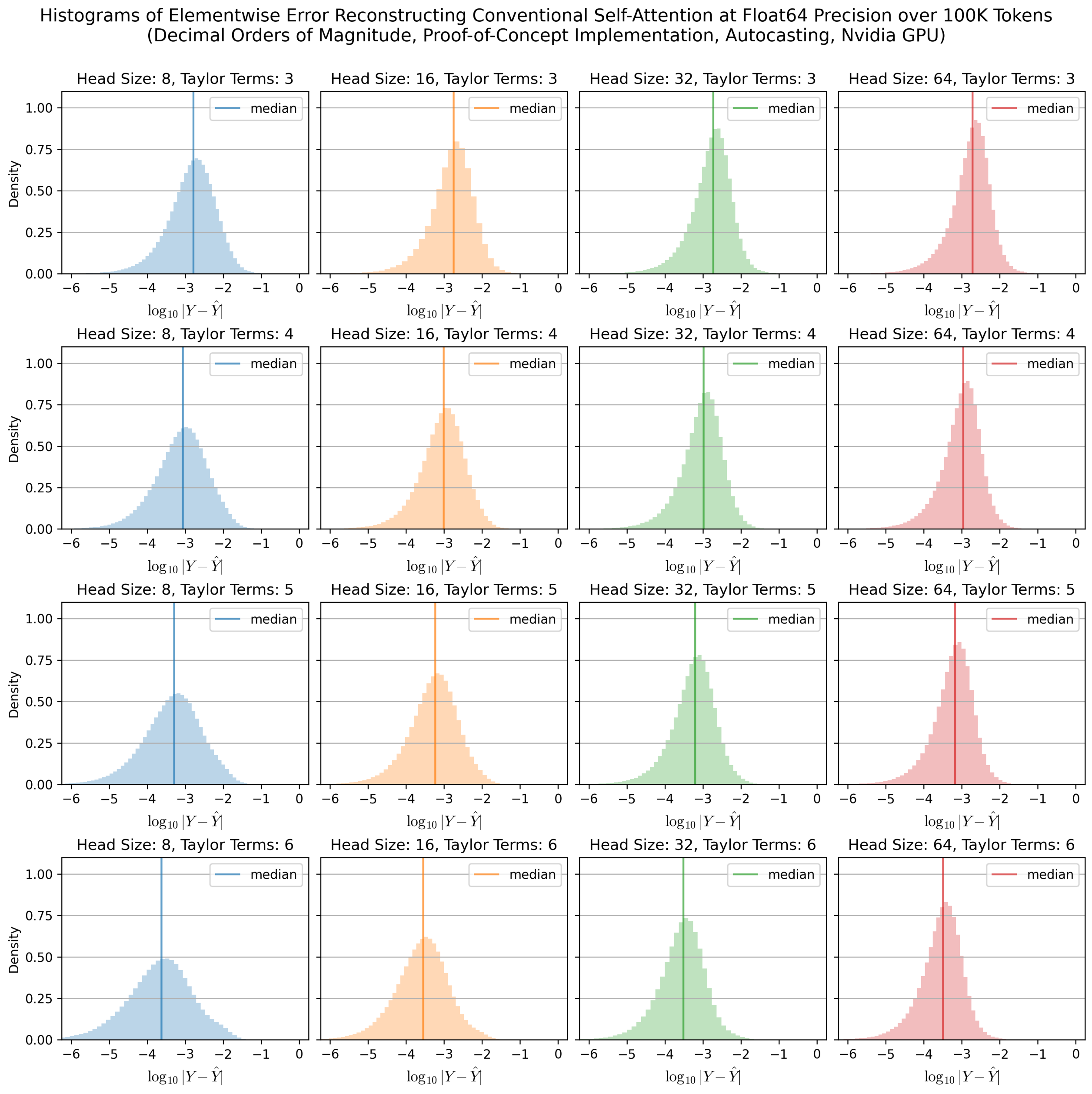
$$Z_T := \sum_{p=0}^{P-1} \left\langle \Phi_p(q_T), Z_{p,T} \right\rangle_{C_p}, \qquad S_T := \sum_{p=0}^{P-1} S_{p,T}^{\top} \Phi_p(q_T).$$

The approximate causal self-attention output can then be written compactly as

$$y_T \approx \frac{S_T}{Z_T}.$$

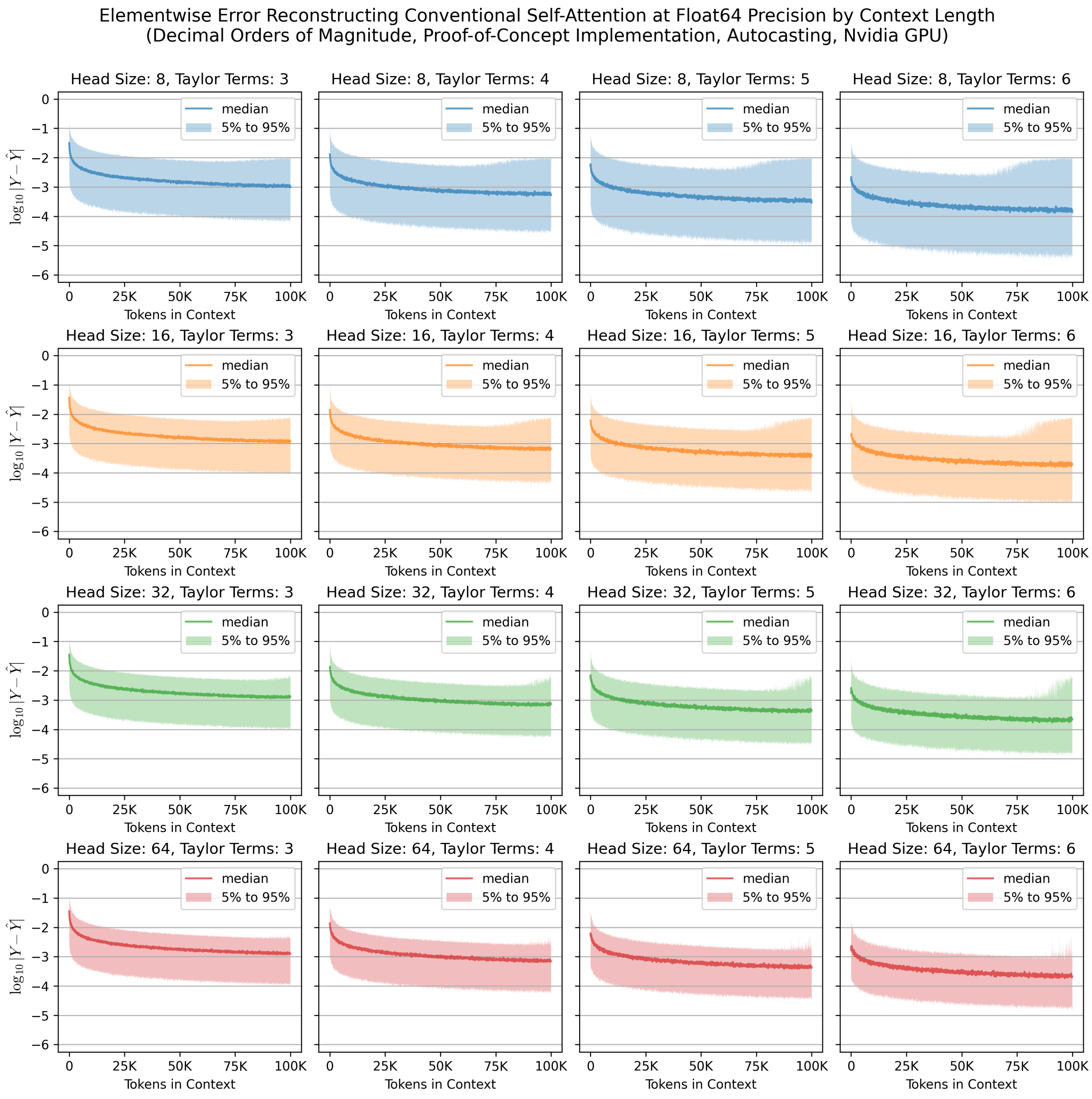
A3 Recovering Conventional Self-Attention

We measure our proof-of-concept implementation's ability to recover conventional self-attention with four different head sizes, $d_K = d_V = d$, and four different Taylor truncation numbers, P, over autoregressive (causal) sequences of 100K tokens, sampling each query, key, and value from $\mathcal{N}(0,1)^d$. We compare the outputs of our implementation, evaluated at auto-casted precision, to those of a conventional implementation, evaluated with the highest-precision floating-point format supported by Nvidia GPUs, Float64, which we treat as "ground truth." We measure elementwise differences in decimal orders of magnitude (*e.g.*, an absolute difference of 0.01 is measured as −2 decimal orders of magnitude).



A4 Recovering Conventional Self-Attention by Token Position

We measure our proof-of-concept implementation's ability to recover conventional self-attention with four different head sizes, $d_K = d_V = d$, and four different Taylor truncation numbers, P, over autoregressive (causal) sequences of 100K tokens, sampling each query, key, and value from $\mathcal{N}(0, 1)^d$. We compare the outputs of our implementation, evaluated at auto-casted precision, to those of a conventional implementation, evaluated with the highest-precision floating-point format supported by Nvidia GPUs, Float64, which we treat as "ground truth." We measure elementwise differences in decimal orders of magnitude (*e.g.*, an absolute difference of 0.01 is measured as −2 decimal orders of magnitude).


Elementwise Error Reconstructing Conventional Self-Attention at Float64 Precision by Context Length
(Decimal Orders of Magnitude, Proof-of-Concept Implementation, Autocasting, Nvidia GPU)
Head Size: 8, Taylor Terms: 3
Head Size: 8, Taylor Terms: 4
Head Size: 8, Taylor Terms: 5
Head Size: 8, Taylor Terms: 6
Head Size: 16, Taylor Terms: 3
Head Size: 16, Taylor Terms: 4
Head Size: 16, Taylor Terms: 5
Head Size: 16, Taylor Terms: 6
Head Size: 32, Taylor Terms: 3
Head Size: 32, Taylor Terms: 4
Head Size: 32, Taylor Terms: 5
Head Size: 32, Taylor Terms: 6
Head Size: 64, Taylor Terms: 3
Head Size: 64, Taylor Terms: 4
Head Size: 64, Taylor Terms: 5
Head Size: 64, Taylor Terms: 6
median
5% to 95%
$\log_{10}|Y - \hat{Y}|$
0
−1
−2
−3
−4
−5
−6
0
25K
50K
75K
100K
Tokens in Context