TRICEPS: TRI-COMPONENT SEPARABLE ATTENTION FOR LONG-CONTEXT PREFILL

Anonymous authors
Paper under double-blind review

ABSTRACT

The $\mathcal{O}(N^2)$ complexity of full attention during prefill in Transformer-based large language models is the main computational bottleneck in applications that require handling long contexts, such as long-document question answering, repository-level code agents or long book summarization. Existing sparse attention methods combine fixed patterns by union (attention sinks, sliding windows, retrieved blocks, vertical and slash patterns), resulting in a mask denser than the mass it captures requires, wasting computation on blocks that carry almost no attention. We instead show that the attention matrix admits a training-free multiplicative *tri-component separable approximation*: a relative-position factor, a key factor and a query-normalization factor. Combined multiplicatively, the three factors induce a score threshold that a union of fixed patterns cannot express. We dynamically evaluate this approximation per head to route computation between three modes: an $\mathcal{O}(N \log N)$ long convolution when the error measured on the last query rows falls below a threshold, an $\mathcal{O}(\rho N^2)$ sparse mode for low-density masks, and delegation to dense or another sparse attention technique otherwise. We evaluate this method on RULER and LongReason with two open-weight long-context models, Qwen3-30B-A3B-Instruct-2507 and Llama-3.1-Nemotron-8B-UltraLong-4M-Instruct, and, at extended context lengths (128k and beyond), we observe speedups in Time-To-First-Token (TTFT) over dense attention while achieving a superior speed–accuracy trade-off compared to existing sparse methods.

1 INTRODUCTION

Efficient long-context processing in Large Language Models (LLMs) is essential to complex generative tasks. However, Transformer models in modern LLM architectures heavily rely on full attention, which suffers from an $\mathcal{O}(N^2)$ computational complexity during prefill. Several research directions mitigate this bottleneck, but so far only sparse attention offers a training-free solution to speed up inference. Sparse attention predicts the pattern of the attention matrix and uses that prediction to avoid computing the negligible parts of the attention. This approach has two intrinsic limitations. First, pattern prediction is imperfect, which forces a conservative over-estimation of the matrix density in order to maintain model accuracy. Second, the true sparsity of the attention matrix does not permit sub-polynomial complexity (Deng et al., 2024).

Jiang et al. (2024) found that a large majority of attention heads has attention matrices exhibiting a vertical-slash pattern. The vertical pattern consists of key tokens that consistently get high or low attention across all query tokens, while the slash pattern consists of bands of adjacent diagonals that assign or suppress attention between tokens at specific relative distances.Jiang et al. (2024); Lai et al. (2025); Xu et al. (2025b) use this observation by sampling a narrow band of attention elements, detecting dominant diagonals and columns within that band, and extrapolating their trajectories across the full attention matrix to construct a block-sparse mask, which ultimately is processed by an efficient block-sparse attention kernel.

Our starting point is also the widespread vertical-slash pattern, but we choose to interpolate rather than extrapolate, by uniformly sampling the attention scores. From the sample we fit a linear model that predicts the *element-wise* vertical-slash pattern directly from the query and key token matrices. The predicted pattern is a separable approximation of the attention, which allows attention computation in *quasi-linear time* or the creation of a block mask that is not merely the union of detected dominant

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

columns and diagonals but the *intersection*. The residual of the fit determines the error of the convolution output, which we measure directly on the last query rows, and the mask built from the same fit exposes the attention mass it would discard. Together with the deployed density, these measurements route each head.

The *tri-component separable approximation* and its quality are computed for each attention head individually during prefill. Thus, we obtain a procedure to speed up prefill while maintaining model accuracy. For high quality approximations, we compute the attention from the approximation directly in $\mathcal{O}(N \log N)$ time, avoiding any additional score computations. Otherwise, when the mask built from the fit can cover the attention mass within the budget, the attention is computed exactly inside the selected blocks. A poor fit ranking indicates that the approximation cannot be trusted, and we delegate the head to dense or another sparse attention technique.

We summarize our contributions as:

- The separable attention approximation and its derivation;
- A three-mode adaptive computation using either convolution, block-sparse, or a delegated method; and
- Empirical speedup with accuracy verification compared to SoTA techniques.

With tensor parallelism among 2 devices and for contexts up to 256k, we achieve on LongReason a $3.4\times$ speed-up in end-to-end Time-To-First-Token (TTFT) with Qwen3-30B-A3B-Instruct-2507, and $2.6\times$ with Llama-3.1-Nemotron-8B-UltraLong-4M-Instruct, at 256k context. Against FlexPrefill, the fastest sparse baseline at this length, the speed-up reaches $1.2\times$ and $1.5\times$, respectively. On RULER, the speed-up in end-to-end TTFT reaches $2.0\times$ over dense attention and $1.1\times$ over FlexPrefill with Qwen3-30B-A3B-Instruct-2507, and $2.0\times$ and $1.3\times$ with Llama-3.1-Nemotron-8B-UltraLong-4M-Instruct.

2 Background

Self-Attention Transformer-based LLMs have an attention mechanism that builds and applies an attention map between all possible token pairs in the input. The attention map is in essence a matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, where N is the number of tokens in the input, i.e. the context length. Concretely, each transformer layer in the model first projects its input $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, D being the embedding size, to a query, key, and value matrix $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N\times d}$ using learnt weight matrices $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v \in \mathbb{R}^{D\times d}$, respectively, for each of the H different heads with dimension d. Afterwards, the query and key matrices also receive a position encoding. In this work, we assume RoPE (Su et al., 2024) which right-multiplies each query and key m with rope matrix $\boldsymbol{R}_m$. Hence, each row m in $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ is defined as $\boldsymbol{q}_m = \boldsymbol{x}_m\boldsymbol{W}_q\boldsymbol{R}_m$, $\boldsymbol{k}_m = \boldsymbol{x}_m\boldsymbol{W}_k\boldsymbol{R}_m$, and $\boldsymbol{v}_m = \boldsymbol{x}_m\boldsymbol{W}_v$, respectively. Finally, it computes the attention as the matrix product

$$\boldsymbol{AV}, \text{ where } \boldsymbol{A} = \text{softmax}\left(\frac{\boldsymbol{QK}^\top - \infty \cdot \boldsymbol{M}}{\sqrt{d}}\right). \tag{1}$$

In prevalent text generating LLMs, which only consist of decoders, $\boldsymbol{M} \in \{0,1\}^{N\times N}$ is an upper-triangular matrix with only 1's in the upper triangle and 0 on the main diagonal. Effectively, each element $A_{m,n}$ in $\boldsymbol{A}$ is defined as

$$A_{m,n} = \frac{\exp\left(\boldsymbol{q}_m\boldsymbol{k}_n^\top/\sqrt{d}\right)}{\sum_{\ell=1}^{m}\exp\left(\boldsymbol{q}_m\boldsymbol{k}_\ell^\top/\sqrt{d}\right)} \text{ if } n \le m \text{ and } 0 \text{ otherwise.}$$

Position Encoding RoPE (Su et al., 2024) encodes the position of each token in the $\boldsymbol{Q}$ and $\boldsymbol{K}$ matrices, so that the Transformer network can also infer meaning from the relative position between tokens. The required arithmetic happens entirely within the real numbers, but the RoPE formulation becomes much simpler if expressed with complex numbers. In this case, we can express $\tilde{\boldsymbol{q}}_m = \boldsymbol{x}_m\tilde{\boldsymbol{W}}_q$ and $\tilde{\boldsymbol{k}}_n = \boldsymbol{x}_n\tilde{\boldsymbol{W}}_k$, where $\tilde{\boldsymbol{W}}_q, \tilde{\boldsymbol{W}}_k \in \mathbb{C}^{D\times d/2}$. The unnormalized attention score for query m on key token n is then expressed as a real-valued scalar $\mathrm{Re}\left[s(m,n)\right]$ where $s(m,n) = \tilde{\boldsymbol{q}}_m\text{diag}(e^{i(m-n)\Theta})\tilde{\boldsymbol{k}}_n^H$ and $\Theta = \left(\theta^{0/d}, \ldots, \theta^{(d-2)/d}\right)$ for some θ.

Attention matrix sparsity patterns Attention matrices are often observed to have a vertical-slash pattern (Jiang et al., 2024; Cheng et al., 2026). That is, most elements in the matrix are close to zero with the exception of only a few columns and diagonals. The dominant columns coincide with high attention across all query tokens for a small number of key tokens, regardless of their position in the context, while the dominant diagonals suggest that at certain relative distances there is consistently high or low attention, regardless of the actual tokens involved. Whereas dominant columns make sense intuitively as important key tokens that have consistently high attention, the origins of the slash pattern are more complex (Cheng et al., 2026).

Retrieval Heads Certain attention heads specialize in particular functions, such as information retrieval (Xiao et al., 2024a; Wu et al., 2024): suppressing them severely impacts the model on those tasks, while restraining other heads does not. We are particularly interested in *retrieval* heads, which look up a *needle* phrase inside a *haystack* of text; a head is considered a retrieval head if it consistently gives its highest attention to the needle at the auto-regressive decode step, when it is copying the needle into the answer.

Retrieval heads in decoder-only models therefore have a characteristic sparsity pattern: because the needle can be anywhere and the causal mask prevents its position from affecting the rows above the question, a retrieval head cannot make the needle a dominant column over the whole matrix when the question comes at the end of the context, nor have similarly dominant diagonals. This does not contradict the transient diagonals of induction heads (Olsson et al., 2022), which start at a certain row and stop a few rows later. Hence, a matrix-wide vertical-slash pattern generally does not occur in retrieval heads.

Block-sparse attention kernels A matrix-vector product $\boldsymbol{A}\boldsymbol{x}$ for $\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{x} \in \mathbb{R}^n$, requires about $2 \times m \times n$ scalar operations. If matrix $\boldsymbol{A}$ is sparse, e.g. only 10% of the elements are non-zero, 90% of the operations can be avoided, allowing for a $10\times$ speed-up. Yet, on contemporary hardware that $10\times$ speed-up can only be realized if these operations are contiguous, because the memory bus and floating point arithmetic cores are mainly optimized for contiguous operations. Only a matrix with block-sparsity, i.e. the zeros and non-zeros always occur in blocks, can be multiplied with the highest efficiency. This makes the block-sparse storage format and computation kernel highly effective for attention computation. Software support for block-sparse matrix formats is present in many machine learning software frameworks, such as PyTorch (Paszke et al., 2019), and in libraries with specialized computational kernels, such as FlashInfer (Ye et al., 2025), while Triton (Tillet et al., 2019) allows building customized kernels.

3 METHOD

3.1 THE TRI-COMPONENT SEPARABLE ATTENTION APPROXIMATION

The core idea of *Triceps* is to approximate attention as a separable product of query-normalization, positional, and key-dependent components.We propose to decompose the attention matrix $\boldsymbol{A}$ as $\boldsymbol{H} \odot \boldsymbol{D} \odot \boldsymbol{C}$, the elementwise product of three matrices $\boldsymbol{H}, \boldsymbol{D}, \boldsymbol{C} \in \mathbb{R}^{N \times N}$, where

- $\boldsymbol{D}$ is lower-triangular and has constant diagonals. I.e. it is a Toeplitz matrix. The element $D_{i,j} = \boldsymbol{d}_{i-j}$ captures the attention attributed by the relative position between key j and query token i. We write the matrix-vector multiplication $\boldsymbol{D}\boldsymbol{x}$ for some vector $\boldsymbol{x} \in \mathbb{R}^N$ as the convolution $\boldsymbol{D}\boldsymbol{x} = \boldsymbol{d} * \boldsymbol{x}$.
- $\boldsymbol{C}$ has constant columns. It is a rank-1 matrix, $\boldsymbol{C} = \boldsymbol{1}\boldsymbol{c}^\top$. The element c_j captures the attention attributed to each key j in the context.
- $\boldsymbol{H}$ has constant rows. It is also a rank-1 matrix, $\boldsymbol{H} = \boldsymbol{h}\boldsymbol{1}^\top$. The element $\boldsymbol{h}_i$ normalizes the attention for query i across all key tokens.

Together, the decomposition gives a *matrix-free* method for the attention

$$\boldsymbol{A}\boldsymbol{V} = (\boldsymbol{H} \odot \boldsymbol{D} \odot \boldsymbol{C})\boldsymbol{V} = \boldsymbol{h} \odot (\boldsymbol{d} * (\boldsymbol{c} \odot \boldsymbol{V})), \tag{2}$$

which also demonstrates that it stores the attention matrix in only $\mathcal{O}(N)$ space. A proof for the equivalence is in Section B.1 in the Appendix.

3.2 Runtime Estimation of the Decomposition

Our main idea is to predict the vertical-slash pattern with a linear model of $\boldsymbol{A}$ in the log-domain.

$$\log \boldsymbol{A}_{ij} = \boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d} - \log \sum_{\ell=1}^{i} \exp\left(\boldsymbol{q}_i \boldsymbol{k}_\ell^\top / \sqrt{d}\right). \tag{3}$$

For instance, we could use the model

$$\boldsymbol{r}_{j-i}\boldsymbol{\alpha} + \boldsymbol{x}_j \boldsymbol{W}_k \boldsymbol{\kappa} \approx \log \boldsymbol{A}_{ij}, \tag{4}$$

where we find $\boldsymbol{\alpha}, \boldsymbol{\kappa} \in \mathbb{R}^d$ using linear least-squares, and our final decomposition would become $\boldsymbol{D}_{i,j} = \exp\left(\boldsymbol{r}_{j-i}\boldsymbol{\alpha}\right)$, $\boldsymbol{C} = \exp\left(\boldsymbol{x}_j \boldsymbol{W}_k \boldsymbol{\kappa}\right)$, and $\boldsymbol{H}_{i,j} = 1$. Here, the $\boldsymbol{r}_{j-i}$ row-vector contains the real parts of the RoPE coefficients

$$\boldsymbol{r}_{j-i} = \left(\cos\left((j-i)\Theta\right)\ \sin\left((j-i)\Theta\right)\right) \text{ where } \Theta = \left(\theta^{\frac{0}{d}}\ \ldots\ \theta^{\frac{d-2}{d}}\right).$$

However, directly computing $\log \sum_{\ell=1}^{i} \exp\left(\boldsymbol{q}_i \boldsymbol{k}_\ell^\top / \sqrt{d}\right)$ is computationally prohibitive, while ignoring this term degrades the quality of the approximation. Indeed, the row-average of $\boldsymbol{q}_i \boldsymbol{k}_j^\top$ over all $j \leq i$ varies significantly between rows i, which introduces a large row-dependent bias in the linear least-squares problem. Furthermore, elements $\boldsymbol{r}_{j-i}\boldsymbol{\alpha}$ and $\boldsymbol{x}_j \boldsymbol{W}_k \boldsymbol{\kappa}$ may have large magnitude, which degrades the approximation quality of $\boldsymbol{D}_{ij}$ and $\boldsymbol{C}_{ij}$ even more.

To address these issues, we centre each predictor around its row-wise mean. Concretely, for each row i we subtract the average RoPE value $\overline{\boldsymbol{r}}_i = \frac{1}{i}\sum_{j=1}^{i} \boldsymbol{r}_{j-i}$ from $\boldsymbol{r}_{j-i}$, the average unroped $\boldsymbol{K}$ value $\boldsymbol{k}_i^{\text{unroped}} = \frac{1}{i}\sum_{j=1}^{i} \boldsymbol{x}_j \boldsymbol{W}_k$ from $\boldsymbol{x}_j \boldsymbol{W}_k$, and the average $\boldsymbol{\mu}_i = \frac{1}{i}\sum_{j=1}^{i} \boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d}$ from $\boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d}$. The final linear model becomes

$$\left(\boldsymbol{r}_{j-i} - \overline{\boldsymbol{r}}_i\right)\boldsymbol{\alpha} + \left(\boldsymbol{x}_j \boldsymbol{W}_k - \boldsymbol{k}_i^{\text{unroped}}\right)\boldsymbol{\kappa} \approx \boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d} - \mu_i \tag{5}$$

Furthermore, we focus the approximation only on the interior of the attention matrix, without the small band of first columns, i.e. the *sink* tokens, and a small band around the main diagonal, i.e. a sliding window. In these two bands, which together are known as the Λ-shape (Xiao et al., 2024b), the distribution of $q_i k_j^\top$ values is atypical and leaving those out increases the quality of the least-squares fit. Concretely, we write the attention matrix as a sum of interior attention matrix and the Λ-shape, i.e. $\boldsymbol{A} = \boldsymbol{A}_{\text{int}} + \boldsymbol{A}_\Lambda$, and we only seek to approximate $\boldsymbol{A}_{\text{int}}$ while we compute $\boldsymbol{A}_\Lambda$ exactly.

To find $\boldsymbol{\alpha}$ and $\boldsymbol{\kappa}$, we compute a small number of the coefficients and solve it as a linear least-squares problem. Unfortunately, the problem is rank-deficient regardless of the number of coefficients we sample. An SVD-based algorithm can handle rank-deficiency and would only require a sample size that is a small multiple of $2d$, i.e. the combined dimensions of $\boldsymbol{\alpha}$ and $\boldsymbol{\kappa}$, to get a solution that is still able to generalize over all positions in the context. However, it is much faster to take about ten times more samples and to apply ridge-regression through addition of a regularisation term, because it allows the use of the highly efficient normal-equations algorithm. Finally, the averages $\overline{\boldsymbol{r}}_i, \boldsymbol{k}_i^{\text{unroped}}$, and μ_i can be computed for all rows simultaneously in $\mathcal{O}(Nd)$ time.

At this point, we have enough to identify the slash pattern $\boldsymbol{D}$ and the vertical pattern $\boldsymbol{C}$. Define the decomposition vector for the slash pattern $\boldsymbol{d} \in \mathbb{R}^{2N-1}$ and the vector for the vertical pattern $\boldsymbol{c} \in \mathbb{R}^N$ as

$$d_{i-j} = \exp\left(\left(\boldsymbol{r}_{j-i} - \overline{\boldsymbol{r}}_N\right)\boldsymbol{\alpha}\right) \text{ for } S < j \leq i - T \text{ and 0 otherwise} \tag{6}$$

$$c_j = \exp\left(\left(\boldsymbol{x}_j \boldsymbol{W}_k - \boldsymbol{k}_N^{\text{unroped}}\right)\boldsymbol{\kappa}\right) \tag{7}$$

then we have

$$D_{i,j}C_{i,j} \approx \exp\left(\boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d} + \delta_i + \lambda_i - \mu_i\right) \text{ for } S < j \leq i - T \text{ and 0 otherwise} \tag{8}$$

where the $\delta_i = \left(\overline{\boldsymbol{r}}_i - \overline{\boldsymbol{r}}_N\right)\boldsymbol{\alpha}$ and $\lambda_i = \left(\boldsymbol{k}_i^{\text{unroped}} - \boldsymbol{k}_N^{\text{unroped}}\right)\boldsymbol{\kappa}$.

To find the horizontal component, consider the equation

$$\mathbf{1} = \boldsymbol{A}\mathbf{1} = (\boldsymbol{A}_{\text{int}} + \boldsymbol{A}_\Lambda)\mathbf{1} = (\boldsymbol{H} \odot \boldsymbol{D} \odot \boldsymbol{C} + \boldsymbol{A}_\Lambda)\mathbf{1}. \tag{9}$$

Solving this for $\boldsymbol{H}$ naively leads to the closed form $\boldsymbol{h} = (\mathbf{1} - \boldsymbol{A}_\Lambda \mathbf{1})/(\boldsymbol{d} * \boldsymbol{c})$. The normalised mass of the sink and sliding-window bands, $\boldsymbol{A}_\Lambda \mathbf{1}$, requires the softmax denominator, which is expensive to compute, i.e. the very quantity $\boldsymbol{h}$ solves for.

The same bands can, however, be computed cheaply in unnormalised form: $A'_{\Lambda,ij} = \exp(\boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d} - \mu_i)$, and in the interior the fit approximates the same exponentiated scores as $\exp(\boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d} - \mu_i) \approx g_i d_{i-j} c_j$ with the row-centring drift $g_i = \exp(-\delta_i - \lambda_i)$. The row mass is therefore known exactly,

$$\boldsymbol{h} = \frac{\mathbf{1}}{\boldsymbol{A}'_\Lambda \mathbf{1} + \boldsymbol{g} \odot (\boldsymbol{d} * \boldsymbol{c})}, \tag{10}$$

and the calibrated convolution output reads

$$\hat{\boldsymbol{A}}\boldsymbol{V} = \boldsymbol{h} \odot \boldsymbol{A}'_\Lambda \boldsymbol{V} + \boldsymbol{h} \odot \boldsymbol{g} \odot \big(\boldsymbol{d} * (\boldsymbol{c} \odot \boldsymbol{V})\big). \tag{11}$$

We call this the convolution-based calibration. Alternatively, a statistics-based calibration estimates the same interior multiplier $\boldsymbol{h}$ through $\boldsymbol{\nu}_i$ s.t.

$$\nu_i \approx \log \sum_{\ell=1}^{i} \exp\left(\boldsymbol{q}_i \boldsymbol{k}_\ell^\top / \sqrt{d} - \mu_i\right), \tag{12}$$

which allows to set $h_i = \exp\left(-\delta_i - \lambda_i - \nu_i\right)$. To estimate ν_i, we rely on the statistical distribution of $\boldsymbol{q}_i \boldsymbol{k}_j^\top$, which is observed to be almost normal. Therefore, we write $\mathrm{z}_i = \boldsymbol{q}_i \boldsymbol{K}^\top$ as the scalar random variable obtained by treating $\boldsymbol{K}$'s rows as samples and we assume that it is almost normal, so that $\mathbb{E}[\exp\left(\mathrm{z}_i - \mu_i\right)] \approx \exp\left(c \cdot \sigma_i^2\right)$ for some constant c, which we fit using ten randomly sampled rows. The row-wise variance $\sigma_i^2 = \mathrm{Var}(\mathrm{z}_i)$ can be computed in $\mathcal{O}(Nd^2)$ time.

All calculations required to find the decomposition as presented above have linear time complexity with respect to the context length N, which makes it viable to compute the decomposition during prefill which would otherwise have quadratic time complexity. Moreover, in the case of grouped query attention (GQA), many calculations involved in solving Equation 5 can be shared inside a KV-group. Nevertheless, the row-wise variance σ^2 that is required by the statistics-based calibration is by far the most costly in terms of raw floating-point operations.

3.3 Quality Assessment and the Three-Mode Adaptive Computation

Following Jiang et al. (2024); Lai et al. (2025), we assess our approximation on the last queries, so that its quality is measured over the full context length. Each head is then assigned one of the three modes by the cascade:

$$mode = \begin{cases} \text{convolution} & \text{if } E_{\text{conv}} \le \tau, \\ \text{block-sparse} & \text{if } M_{\text{drop}} \le 1 - \exp(-k_c \tau) \ \text{ and } \ \rho \le \rho_0, \\ \text{delegation} & \text{otherwise,} \end{cases} \tag{13}$$

evaluated in this order, so convolution takes priority whenever $E_{\text{conv}} \le \tau$. Here E_{conv} and M_{drop} are the two measures defined below and ρ is the deployed density of the block-sparse mask. For each fixed global choice of k_c and the hardware-dependent ρ_0, τ is the per-deployment quality-control parameter. The constant k_c maps τ to the allowed dropped-mass budget, $1 - \exp(-k_c \tau)$. We hold k_c fixed when sweeping τ.

Convolution mode In this mode the attention output is computed directly from the tri-component fit, without materializing the interior of the attention matrix. With $\boldsymbol{A}_{\text{int}} = \boldsymbol{H} \odot \boldsymbol{D} \odot \boldsymbol{C}$, where $H_{i,j} = \boldsymbol{h}_i \boldsymbol{g}_i$ is the drift-corrected multiplier, $D_{i,j} = \boldsymbol{d}_{i-j}$ and $C_{i,j} = \boldsymbol{c}_j$, the interior is applied matrix-free through the convolution of Equation 2, read row-wise:

$$\left[\boldsymbol{A}_{\text{int}} \boldsymbol{V}\right]_{i,:} = \boldsymbol{h}_i g_i \left(\boldsymbol{d} * (\boldsymbol{c} \odot \boldsymbol{V})\right)_i,$$

and the band output is added back:

$$\boldsymbol{O}^{\text{conv}} = \boldsymbol{h} \odot \big(\boldsymbol{A}'_\Lambda \boldsymbol{V} + \boldsymbol{g} \odot (\boldsymbol{d} * (\boldsymbol{c} \odot \boldsymbol{V}))\big). \tag{14}$$

The product with the Λ-shape $\boldsymbol{A}'_\Lambda \boldsymbol{V}$ is computed in $\mathcal{O}(N)$ time, because $\boldsymbol{A}'_\Lambda$ has a constant number of non-zeros per row. The convolution in the interior part has the highest time complexity, but the attention output computation of this mode is still only $\mathcal{O}(N \log N)$, since the convolution theorem allows it to be expressed with $2d + 1$ fast Fourier transforms.

Convolution has no protection against outlier rows: a single badly fitted row is the failure mode. We therefore only enter this mode when the ℓ_2 error is bounded on the last k queries:

$$E_{\text{conv}} = \max_{N-k+1 \le i \le N} \frac{\left\| \boldsymbol{O}_i^{\text{conv}} - \boldsymbol{O}_i^{\text{dense}} \right\|_2}{\left\| \boldsymbol{O}_i^{\text{dense}} \right\|_2}, \tag{15}$$

where $\boldsymbol{O}^{\text{dense}}$ is the exact attention output, computed as full attention on those k rows. Appendix B.2 shows that this error is exactly the covariance between the fit residuals and the values; as the scores alone are not enough, we measure the quality of the convolution directly on the attention output. We use the convolution mode if $E_{\text{conv}} \le \tau$, where τ is the error budget. The convolution mode targets long contexts: its per-head cost grows slowly and it is the cheapest at 256k (Figure 2).

Block-sparse mode This mode is more permissive than Convolution: the fitted decomposition no longer computes the attention output directly, but only locates the blocks that carry attention mass; the scores, and ultimately the output, are then computed exactly from the $\boldsymbol{Q}\boldsymbol{K}^\top$ product inside those blocks. To this end, we aggregate the decomposition at block granularity (blocks of B tokens) in the log-domain: for each query block m and key block n we take the maximum of $\log \boldsymbol{h}_i$ (the statistics-calibrated multiplier $\boldsymbol{h}$ of Equation 12) and of $\log c_j$ over the block, and we average the block-wise maximum of $\log d_t$ over adjacent offset blocks. Since the calibration absorbs the row-length normalization into ν_i, we add back $\log(m+1)$ so that the score is comparable across rows and the threshold is relative to the uniform distribution rather than an absolute attention probability. The resulting block score is

$$\widetilde{\boldsymbol{A}}_{\text{block}}[m, n] = \widetilde{h}_m + \widetilde{d}_{m-n} + \widetilde{c}_n + \log(m+1). \tag{16}$$

The mask keeps every block whose block score reaches a single threshold θ that is shared by the whole head, together with every block of the Λ-shape included,

$$(m, n) \in \mathcal{M} \quad \Longleftrightarrow \quad \widetilde{\boldsymbol{A}}_{\text{block}}[m, n] \ge \theta. \tag{17}$$

Because the score is additive in the log-domain, this rule makes the mask a *multiplicative* combination of the three fitted components: a strong vertical component can compensate for a weak diagonal one, and vice versa, so the mask can follow attention patterns that no single component alone would select. Furthermore, $\widetilde{h}_m$ adapts the mask to each query block's query–key distribution; both mechanisms let Triceps reach sparser masks.

We find θ on the same last k rows as E_{conv}: for each of these rows i, we let b_i be the number of blocks that the row keeps, i.e. its Λ-shape blocks plus the shortest prefix of interior blocks, ordered by decreasing $\widetilde{\boldsymbol{A}}_{\text{block}}$, whose attention mass covers all but $1 - \exp(-k_c\tau)$ of the row's mass. Here the fit provides the ordering of the blocks while counting b_i, and the true attention only measures the mass they carry. We set θ to the largest level whose mask keeps at least the block budget of the measured rows, extrapolated from these k rows; resolving the threshold θ evaluates the causal block triangle at a cost of $\mathcal{O}((N/B)^2)$ coarse scalar block operations (i.e., quadratic only in the number of blocks; Table 6),

$$\left|\mathcal{M}\right| \;\ge\; \frac{\lceil N/B \rceil + 1}{2k} \sum_i b_i, \tag{18}$$

where $\mathcal{M}$ is the block mask, $|\mathcal{M}|$ the number of blocks it keeps, and the sum runs over the k measured rows. Unlike Convolution, whose failure mode is either an under- or over-estimated attention weight, this mode fails by discarding blocks with high attention mass, so we deploy the mask only if the mass it discards on the last k rows satisfies

$$M_{\text{drop}} = 1 - \frac{1}{k} \sum_i \sum_{j \in \mathcal{M}(i)} A_{i,j} \;\le\; 1 - \exp(-k_c\tau), \tag{19}$$

and we verify that its density, computed directly from the block scores without materializing the mask, satisfies $\rho \le \rho_0$. Here, $\mathcal{M}(i)$ denotes the set of keys that the mask keeps for row i, M_{drop} is

the fraction of attention mass the mask discards on these rows, and ρ_0 is the density above which dense attention is cheaper than the block-sparse kernel on our hardware. A head that fails either check is delegated instead. This occurs either if the head is naturally dense, or when the fit ranks the blocks incorrectly,which typically happens for retrieval heads (Wu et al., 2024).

Delegation mode If neither of the two conditions above holds, i.e. $E_{\text{conv}} > \tau$ and ($M_{\text{drop}} > 1 - \exp(-k_c\tau)$ or $\rho > \rho_0$), we delegate the head's attention computation: either FA2 or any other sparse attention method can be used.

4 Related Work

Sparse attention methods Sparsity in the attention matrix enables acceleration of both inference stages. *Decode* acceleration through sparsity is demonstrated by StreamingLLM (Xiao et al., 2024b), which keeps only the Λ-shape – sink tokens and sliding window – in the kv-cache. DuoAttention (Xiao et al., 2024a) restricts this eviction scheme to streaming heads and retains full attention on retrieval heads. RazorAttention (Tang et al., 2024a) applies a similar head-categorization to its eviction policy. We share this retrieval/non-retrieval distinction with both, but identify it at runtime rather than via an offline phase. Nevertheless, even the non-retrieval heads function can be impacted by KV-cache eviction. When storage permits, Quest (Tang et al., 2024b) avoids eviction altogether and targets speed alone.

Our work targets *prefill* acceleration, like MInference (Jiang et al., 2024), the first to exploit the vertical-slash pattern, and FlexPrefill (Lai et al., 2025) the first to allocate computational budget per head dynamically. Both also distinguish between vertical-slash and non-vertical-slash heads; offline for MInference, online for FlexPrefill. XAttention (Xu et al., 2025b) and PRISM (Wang et al., 2026a) acknowledge head-level pattern variation but treat all heads uniformly. PBS-Attn (Wang et al., 2026b) takes a different route and permutes keys within segments so that the mass-carrying attention weights cluster into the same blocks: fewer blocks then cover the same attention mass, making heads sparser. These methods are training-free in the sense that their offline calibration adjusts a small number of hyperparameters by processing a small dataset, in contrast to methods with a full training phase such as SeerAttention (Gao et al., 2025).

Structural properties of pretrained transformers Structure emerges spontaneously in trained Transformer-based LLMs, both in feed-forward layers Mirzadeh et al. (2023) and in attention. Our method exploits two such structures: the retrieval/non-retrieval head classification of Wu et al. (2024), and the vertical-slash pattern analyzed by Cheng et al. (2026).

Linear/sub-quadratic attention approaches Sub-quadratic attention mechanisms generally cannot be retrofitted to pretrained LLMs without retraining. LoLCATS (Zhang et al., 2025) articulates this ambition very well, however, but still requires training millions of parameter weights, just like Lizard (Nguyen et al., 2026). Once global calibration constants are chosen, only the error budget τ is adjusted per deployment. Long-convolution models (Qin et al., 2023; Poli et al., 2023), require pretraining and still face quadratic costs in auto-regressive generation (Massaroli et al., 2023).

5 Experiments

5.1 Experimental Setup

Implementation. We implement *Triceps* as a custom prefill only attention backend in vLLM v0.17.0. The block-sparse masks are executed by a custom Triton kernel, and the convolution is computed with the FFT implementation of torchaudio. The decomposition is fitted by ridge-regularised least squares on up to 10^4 sampled interior score pairs, and the routing statistics are measured on the last $k = 64$ query rows. We use Lai et al. (2025) as the delegated attention method and disable full attention, so a head that leaves the convolution or block-sparse path is always delegated to FlexPrefill. The competitors are implemented from their released code and run with their reference configurations. MInference requires an offline calibration pass that we ran with its original search (vertical–slash heads, 1000 verticals and 6096 slashes). As its budget is fixed offline, it does not adapt as the context grows and its accuracy degrades markedly, up to 14 points below

dense at 256k; we therefore report it separately in Appendix A.3 rather than merging it into the main comparison. Section C lists the algorithms and Table 6 their leading-order FLOP counts.

Models and benchmarks. We evaluate on Qwen3-30B-A3B-Instruct-2507 Yang et al. (2025) and Llama-3.1-Nemotron-8B-UltraLong-4M-Instruct (Xu et al., 2025a), for which we use the shorthands Qwen3 and Nemotron respectively. Accuracy and TTFT are measured on LongReason (Ling et al., 2025) and RULER (Hsieh et al., 2024) at 64k, 128k and 256k context. RULER is averaged over its 13 subtasks with 100 prompts each, and LongReason over its 794 prompts; beyond 128k we extend the LongReason prompts with C4 Raffel et al. (2020) distractors, following the method of the benchmark authors. Decoding is computed densely.

Protocol. Accuracies are means over prompts; each TTFT point is the median over 10 timing prompts. TTFT is measured end-to-end on the full model: it includes the entire pipeline up to the generation of the first token, including the overhead of each method and the attention kernels. Timings are taken after a warmup, with 10 prompts per point; the RULER timing uses `niah_multikey_2`. All timing points use TP=2, for the baselines and for Triceps alike. The submitted configuration uses $\tau = 0.4$, block size $B = 128$, sink width $S = 30$ and window width $T = 96$. We use a dropped-mass slope $k_c = 0.5$ and a density threshold $\rho_0 = 0.60$. All main results use this configuration; Appendix A.1 studies separately a lower fixed k_c, with τ as the only swept parameter.

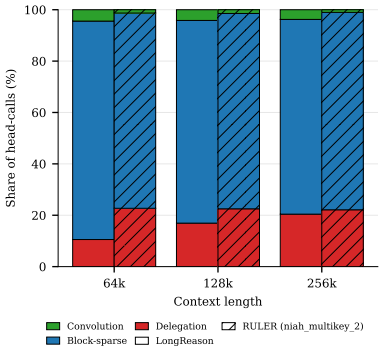

Figure 1: Routing shares for Nemotron on LongReason and RULER.

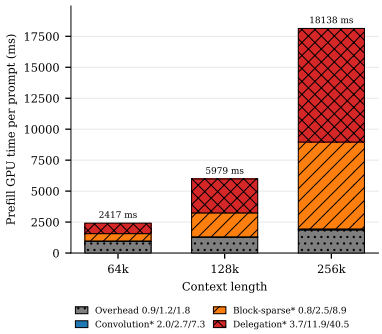

Figure 2: Prefill cost breakdown by routed path (Nemotron, RULER).

5.2 Triceps on different long context benchmarks

LongReason is of particular interest since it allows sampling prompts of arbitrary length while the tasks themselves are realistic: it mixes multi-step reasoning questions with realistic filler text, a pattern that naturally occurs in chatbot–user dialogues. RULER instead focuses on pure information retrieval. The two benchmarks exercise the routing differently: the separable approximation captures fewer of the retrieval-heavy RULER heads, which the routing detects from the measured tail-row error, so the delegated share is roughly twice as large as on LongReason at 64k (23% vs 11%). Figure 1 shows these shares per benchmark and context length; Table 1 reports the accuracy and Figure 3 the TTFT speed-ups (absolute times in Table 5), all with the same configuration.

Figure 2 breaks the prefill GPU time down by routed path for Nemotron on the RULER `niah_multikey_2` task (mean over 100 prompts; host, idle and decoding excluded). The decomposition and routing overhead is amortised, from 39% of the GPU time at 64k to 10% at 256k. The paths scale as intended: convolution is the fastest per routed head at 256k (2.0 to 7.3 ms), while block-sparse and delegation costs grow by an order of magnitude per head for a $4\times$ longer context (0.8 to 8.9 and 3.7 to 40.5 ms). FlexPrefill's own overhead, in particular its selection pass, is included in the delegation cost; the timings cover the heads the router selected and therefore describe the entire routed workload, not a single kernel benchmark. Table 2 reports the corresponding routing shares and deployed densities: on LongReason the all-paths density stays below 15% at every tested length, whereas the retrieval-heavy RULER tasks delegate up to a third of the head-calls and reach densities of about 25%.

Table 1: Score on various context lengths. The full RULER detail is in Table 4; Triceps rows use $k_c = 0.5$, $\tau = 0.4$.

(a). LongReason

Method	64k	128k	256k
Qwen-3-30B-A3B			
Dense	76.57%	73.43%	73.30%
FlexPrefill	**76.01%**	71.91%	72.29%
PBS-Attn	75.94%	72.41%	**73.68%**
PRISM	75.81%	**74.06%**	73.30%
Triceps ($\tau = 0.4$)	67.62%	70.15%	67.51%
Llama-3.1-Nemotron-8B-UltraLong-4M			
Dense	45.97%	45.66%	45.10%
FlexPrefill	45.80%	**44.96%**	44.20%
PBS-Attn	44.30%	43.83%	43.80%
PRISM	45.50%	44.90%	**45.00%**
Triceps ($\tau = 0.4$)	**46.22%**	43.58%	43.32%

(b). RULER, full result in Table 4.

Method	64k	128k	256k
Qwen-3-30B-A3B			
Dense	92.85%	89.46%	84.82%
FlexPrefill	89.61%	86.15%	**84.57%**
PBS-Attn	89.73%	85.06%	82.04%
PRISM	**90.54%**	**86.43%**	82.96%
Triceps ($\tau = 0.4$)	88.78%	85.22%	82.63%
Llama-3.1-Nemotron-8B-UltraLong-4M			
Dense	75.74%	71.58%	65.54%
FlexPrefill	73.38%	70.64%	62.16%
PBS-Attn	72.49%	71.06%	61.74%
PRISM	**73.58%**	**71.17%**	**62.69%**
Triceps ($\tau = 0.4$)	72.33%	69.57%	60.29%

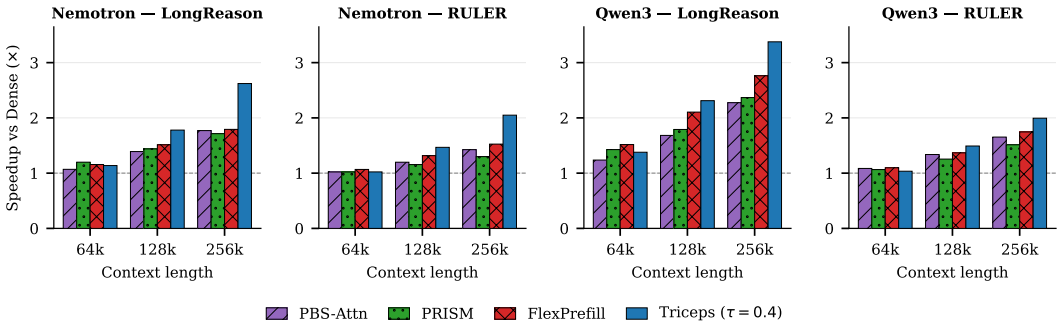

Figure 3: TTFT speed-up over dense attention (dashed) for Nemotron (left) and Qwen3 (right) on LongReason and RULER; Triceps uses $k_c = 0.5$, $\tau = 0.4$, absolute times in Table 5.

6 Limitations

Routing measures approximation error and dropped mass on sampled tail rows; it does not guarantee end-to-end accuracy. Appendix A.1 examines sensitivity to the global slope k_c on four selected RULER subtasks; generalization to the full benchmark and LongReason remains to be established. The optimal τ depends on the task's sensitivity to perturbations, so a workload whose answers degrade under small perturbations may require a more conservative τ.

A second limitation is scheduling: the block-sparse and delegated paths have a head-dependent cost while convolution is uniform, so heads in a layer finish at different times, and the imbalance reduces the attainable speedup once the KV cache spreads heads across devices; we leave load balancing and work stealing to future work. Finally, Triceps accelerates prefill only; decoding runs densely, and the speedups are modest below 128k, where the decomposition and routing overhead is not yet amortised.

7 Conclusion

In this work, we introduced a tri-component separable attention approximation that exploits the vertical-slash patterns occurring in attention heads. Triceps demonstrates that a substantial fraction of attention heads can be approximated as separable operators, enabling sub-quadratic attention computation without retraining. By employing a runtime quality measurement to switch between three methods of attention computation, Triceps achieves speedups up to $3.4\times$ at 256k compared to dense attention.

AI USE STATEMENT

We used AI tools to draft parts of the paper, to review and critique the content, and to assist with the efficiency of our implementation. The research idea, the novelty of the technique, the method design and the experimental decisions are the authors' own; the AI tools served as assistance only. All AI-assisted work has been reviewed by the authors, who take full responsibility for the final content.

REPRODUCIBILITY STATEMENT

All algorithms are described in Appendix C, and the experimental protocol, models, benchmarks and hyperparameters are detailed in Section 5. We provide anonymized code and configuration files to reproduce the reported speedups and accuracies, and we will open-source the repository upon acceptance.

REFERENCES

Lawrence D. Brown. *Fundamentals of Statistical Exponential Families with Applications in Statistical Decision Theory*, volume 9 of *Lecture Notes-Monograph Series*. Institute of Mathematical Statistics, 1986. URL http://www.jstor.org/stable/4355554.

Yuan Cheng, Fengzhuo Zhang, Yunlong Hou, Cunxiao Du, Chao Du, Tianyu Pang, Aixin Sun, and Zhuoran Yang. Demystifying the slash pattern in attention: The role of rope, 2026. URL https://arxiv.org/abs/2601.08297.

Yichuan Deng, Zhao Song, Jing Xiong, and Chiwun Yang. How sparse attention approximates exact attention? your attention is naturally n^c-sparse. *arXiv preprint arXiv:2404.02690*, 2024.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. Seerattention: Learning intrinsic sparse attention in your llms, 2025. URL https://arxiv.org/abs/2410.13276.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference, 2025. URL https://arxiv.org/abs/2502.20766.

Zhan Ling, Kang Liu, Kai Yan, Yifan Yang, Weijian Lin, Ting-Han Fan, Lingfeng Shen, Zhengyin Du, and Jiecao Chen. Longreason: A synthetic long-context reasoning benchmark via context expansion, 2025. URL https://arxiv.org/abs/2501.15089.

Stefano Massaroli, Michael Poli, Dan Fu, Hermann Kumbong, Rom Parnichkun, David Romero, Aman Timalsina, Quinn McIntyre, Beidi Chen, Atri Rudra, et al. Laughing hyena distillery: Extracting compact recurrences from convolutions. *Advances in Neural Information Processing Systems*, 36:17072–17116, 2023.

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models, 2023. URL https://arxiv.org/abs/2310.04564.

Chien Van Nguyen, Huy Nguyen, Ruiyi Zhang, Hanieh Deilamsalehy, Puneet Mathur, Viet Dac Lai, Haoliang Wang, Jayakumar Subramanian, Ryan A. Rossi, Trung Bui, Nikos Vlassis, Franck Dernoncourt, and Thien Huu Nguyen. Lizard: An efficient linearization framework for large language models, 2026. URL https://arxiv.org/abs/2507.09025.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022. URL https://arxiv.org/abs/2209.11895.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, pp. 28043–28078. PMLR, 2023.

Zhen Qin, Xiaodong Han, Weixuan Sun, Bowen He, Dong Li, Dongxu Li, Yuchao Dai, Lingpeng Kong, and Yiran Zhong. Toeplitz neural network for sequence modeling, 2023. URL https://arxiv.org/abs/2305.04749.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

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

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads, 2024a. URL https://arxiv.org/abs/2407.15891.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024b. URL https://arxiv.org/abs/2406.10774.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Xinghao Wang, Pengyu Wang, Xiaoran Liu, Fangxu Liu, Jason Chu, Kai Song, and Xipeng Qiu. Prism: Spectral-aware block-sparse attention, 2026a. URL https://arxiv.org/abs/2602.08426.

Xinghao Wang, Pengyu Wang, Dong Zhang, Chenkun Tan, Shaojun Zhou, Zhaoxiang Liu, Shiguo Lian, Fangxu Liu, Kai Song, and Xipeng Qiu. Sparser block-sparse attention via token permutation, 2026b. URL https://arxiv.org/abs/2510.21270.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads, 2024a. URL https://arxiv.org/abs/2410.10819.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024b. URL https://arxiv.org/abs/2309.17453.

Chejian Xu, Wei Ping, Peng Xu, Zihan Liu, Boxin Wang, Mohammad Shoeybi, and Bryan Catanzaro. From 128k to 4m: Efficient training of ultra-long context large language models. *arXiv preprint*, 2025a.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring, 2025b. URL https://arxiv.org/abs/2503.16428.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving, 2025. URL https://arxiv.org/abs/2501.01005.

Michael Zhang, Simran Arora, Rahul Chalamala, Alan Wu, Benjamin Spector, Aaryan Singhal, Krithik Ramesh, and Christopher Ré. Lolcats: On low-rank linearizing of large language models, 2025. URL https://arxiv.org/abs/2410.10254.

A MORE EXPERIMENTS AND ABLATION STUDIES

A.1 SENSITIVITY TO THE ERROR BUDGET τ

We study the sensitivity to τ with a lower and fixed global calibration constant, $k_c = 0.4$, on four RULER subtasks of Llama-3.1-Nemotron-8B-UltraLong-4M-Instruct at 256k: `niah_multikey_2`, `niah_multiquery`, `vt` and `niah_single_1`. Only τ varies, over $\{0.35, 0.4, 0.45, 0.5, 0.55, 0.6\}$, with k_c held at 0.4 and ρ_0 unchanged; we compare against the submitted configuration ($k_c = 0.5$, $\tau = 0.4$) on the same prompts. Figure 4 reports the accuracy and the routed shares, and Figure 5 the per-head modes at $\tau = 0.4$ and $\tau = 0.6$. The subtasks differ in sensitivity: `niah_single_1` maintains its accuracy across the sweep, while `niah_multikey_2` and `vt` degrade as τ grows.

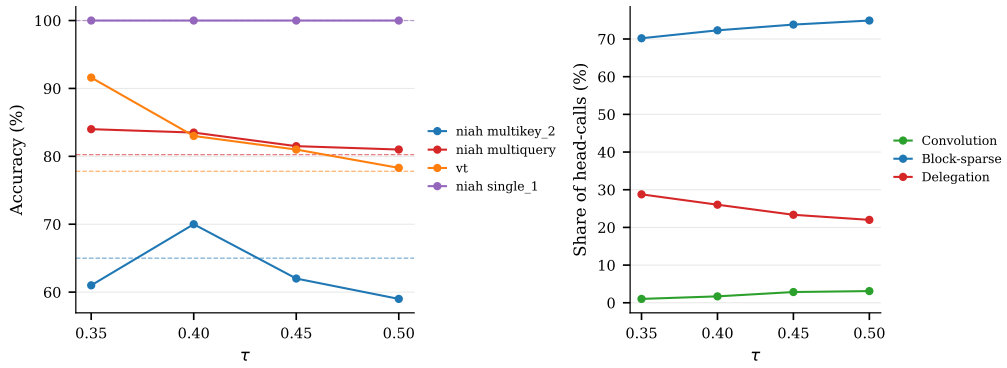

Figure 4: Accuracy (left) and share of head-calls per routed path (right) versus τ at $k_c = 0.4$ on the four subtasks of Section A.1 (Nemotron, 256k, 100 prompts). Dashed lines mark the submitted configuration ($k_c = 0.5$, $\tau = 0.4$) on the same prompts; the right panel averages the shares over the subtasks.

A.2 ROUTING SHARES AND DEPLOYED DENSITIES

Table 2 reports, for the submitted configuration ($k_c = 0.5$, $\tau = 0.4$), the fraction of heads routed to each of the three attention paths and the resulting deployed densities. Because τ is set conservatively, the convolution path is chosen for a small minority of heads (2–14%), the block-sparse path carries the majority (59–85%), and delegation grows with context length. The all-paths density remains between 11% and 25% on both benchmarks; on LongReason it stays below 15% at every tested length.

Table 2: Routing shares (%) and deployed densities for Triceps ($k_c = 0.5$, $\tau = 0.4$): (a) over LongReason (794 prompts), (b) averaged over the 13 RULER subtasks.

(a). LongReason (794 prompts)

Statistic	64k	128k	256k
Qwen-3-30B-A3B			
Convolution	13.5%	9.6%	6.1%
Block-sparse	68.9%	70.4%	72.6%
Delegation	17.6%	20.0%	21.3%
Block-sparse only density	3.3%	2.6%	2.0%
All-paths density	12.7%	11.9%	11.0%
Llama-3.1-Nemotron-8B-UltraLong-4M			
Convolution	4.4%	4.2%	3.8%
Block-sparse	85.0%	78.9%	75.8%
Delegation	10.5%	16.9%	20.4%
Block-sparse only density	5.5%	5.5%	5.7%
All-paths density	13.0%	14.4%	14.2%

(b). RULER (average of the 13 subtasks)

Statistic	64k	128k	256k
Qwen-3-30B-A3B			
Convolution	9.7%	6.9%	4.6%
Block-sparse	59.0%	61.5%	63.6%
Delegation	31.2%	31.6%	31.8%
Block-sparse only density	12.0%	8.8%	7.1%
All-paths density	25.1%	21.9%	19.4%
Llama-3.1-Nemotron-8B-UltraLong-4M			
Convolution	2.2%	2.0%	1.7%
Block-sparse	75.8%	73.7%	72.9%
Delegation	22.0%	24.4%	25.4%
Block-sparse only density	12.0%	12.1%	11.7%
All-paths density	22.9%	23.4%	22.3%

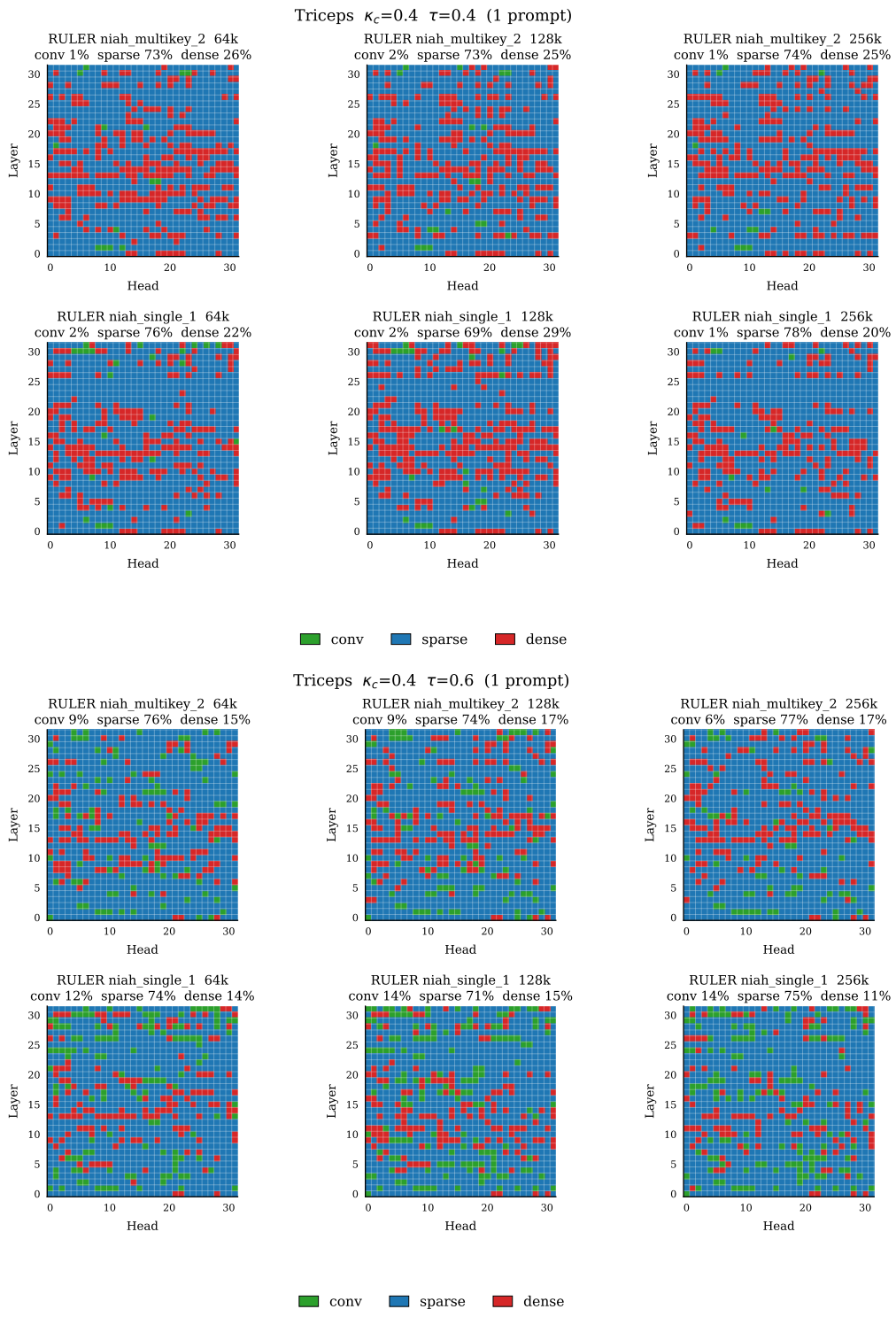

Figure 5: Mode maps (single prompt) for Nemotron at $k_c = 0.4$: (top) $\tau = 0.4$ and (bottom) $\tau = 0.6$, over the RULER subtasks `niah_multikey_2` and `niah_single_1` at 64k, 128k and 256k. Green, blue and red mark heads routed to convolution, block-sparse attention and delegation (dense); the delegated bucket executes FlexPrefill. Raising τ increases the convolution share (from 1–2% at $\tau = 0.4$ to 6–14% at $\tau = 0.6$) and reduces delegation.

A.3 MINFERENCE REFERENCE RESULTS

MInference fixes its vertical–slash pattern offline, so its sparsity budget does not adapt as the context grows; its accuracy therefore falls up to 14 points below Dense at 256k (Nemotron on RULER). For this reason it is not part of the online comparison, and its accuracy and TTFT are reported next to the Dense baseline in Table 3.

Table 3: MInference and Dense reference results: accuracy (average over the 13 RULER subtasks and over LongReason) and TTFT (LongReason and RULER `niah_multikey_2`), with TP=2.

			64k		128k		256k	
Metric	**Model**	**Method**	**LR**	**RULER**	**LR**	**RULER**	**LR**	**RULER**
Accuracy (%)	Qwen3-30B-A3B	Dense	76.57%	92.85%	73.43%	89.46%	73.30%	84.82%
		MInference	74.56%	91.63%	70.30%	87.46%	64.99%	81.87%
	Nemotron-8B	Dense	45.97%	75.74%	45.66%	71.58%	45.10%	65.54%
		MInference	42.56%	69.76%	42.30%	60.86%	41.20%	51.12%
TTFT (s)	Qwen3-30B-A3B	Dense	6.16s	6.11s	22.67s	22.57s	80.06s	79.99s
		MInference	5.61s	6.28s	11.65s	13.95s	24.34s	29.49s
	Nemotron-8B	Dense	5.25s	5.23s	17.48s	17.43s	63.79s	63.79s
		MInference	4.39s	4.81s	8.76s	10.02s	18.64s	22.07s

A.4 FULL EVALUATION RESULTS

Table 4: Full accuracy detail for RULER, for 64k, 128k and 256k context size.

	Method	niah single1	niah single2	niah single3	niah multikey1	niah multikey2	niah multikey3	niah multivalue	niah multiquery	vt	cwe	fwe	qa1	qa2	avg
	Qwen-3-30B-A3B														
64k	Dense	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	98.75%	99.75%	100.00%	69.00%	97.67%	84.00%	58.00%	92.85%
	FlexPrefill	100.00%	100.00%	99.00%	98.00%	100.00%	99.00%	95.00%	99.75%	88.40%	57.10%	95.67%	80.00%	53.00%	89.61%
	PBS-Attn	100.00%	100.00%	100.00%	100.00%	98.00%	99.00%	93.75%	99.75%	73.60%	78.70%	91.67%	75.00%	57.00%	89.73%
	PRISM	100.00%	100.00%	100.00%	99.00%	99.00%	98.00%	94.25%	100.00%	77.00%	84.10%	91.67%	78.00%	56.00%	**90.54%**
	Triceps ($\tau = 0.4$)	100.00%	100.00%	100.00%	100.00%	98.00%	100.00%	92.00%	100.00%	79.20%	64.60%	93.33%	78.00%	49.00%	88.78%
	Llama-3.1-Nemotron-8B-UltraLong-4M														
	Dense	100.00%	100.00%	100.00%	86.00%	94.00%	53.00%	84.50%	98.75%	75.20%	7.50%	85.67%	59.00%	41.00%	75.74%
	FlexPrefill	100.00%	97.00%	100.00%	78.00%	89.00%	49.00%	89.25%	96.25%	78.60%	4.20%	86.66%	47.00%	39.00%	73.38%
	PBS-Attn	99.00%	99.00%	100.00%	80.00%	86.00%	37.00%	84.75%	92.75%	76.80%	9.40%	82.66%	59.00%	36.00%	72.49%
	PRISM	100.00%	98.00%	99.00%	84.00%	89.00%	35.00%	87.50%	96.75%	78.80%	5.50%	83.00%	58.00%	42.00%	**73.58%**
	Triceps ($\tau = 0.4$)	100.00%	99.00%	97.00%	73.00%	77.00%	51.00%	86.50%	95.75%	70.00%	20.40%	91.67%	42.00%	37.00%	72.33%
	Qwen-3-30B-A3B-Instruct-2507														
128k	Dense	100.00%	100.00%	100.00%	98.00%	100.00%	100.00%	98.50%	97.50%	100.00%	59.70%	84.33%	76.00%	49.00%	89.46%
	FlexPrefill	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	99.25%	99.75%	84.00%	21.30%	94.66%	72.00%	49.00%	86.15%
	PBS-Attn	100.00%	100.00%	100.00%	98.00%	96.00%	96.00%	96.50%	99.25%	57.00%	53.70%	93.33%	66.00%	50.00%	85.06%
	PRISM	100.00%	100.00%	100.00%	99.00%	96.00%	97.00%	98.00%	99.50%	59.00%	64.10%	94.00%	65.00%	52.00%	**86.43%**
	Triceps ($\tau = 0.4$)	100.00%	100.00%	100.00%	100.00%	99.00%	98.00%	98.50%	100.00%	70.80%	31.60%	95.00%	66.00%	49.00%	85.22%
	Llama-3.1-Nemotron-8B-UltraLong-4M														
	Dense	100.00%	95.00%	100.00%	83.00%	87.00%	17.00%	85.75%	94.00%	85.80%	5.60%	78.33%	59.00%	40.00%	71.58%
	FlexPrefill	100.00%	92.00%	97.00%	75.00%	83.00%	38.00%	90.00%	92.25%	86.80%	4.30%	77.99%	47.00%	35.00%	70.64%
	PBS-Attn	100.00%	92.00%	100.00%	80.00%	86.00%	20.00%	84.75%	92.75%	90.40%	8.20%	76.67%	55.00%	38.00%	71.06%
	PRISM	99.00%	94.00%	100.00%	85.00%	85.00%	29.00%	85.75%	92.25%	89.80%	1.70%	72.67%	53.00%	38.00%	**71.17%**
	Triceps ($\tau = 0.4$)	100.00%	90.00%	97.00%	76.00%	79.00%	37.00%	88.50%	93.00%	74.20%	6.70%	82.00%	44.00%	37.00%	69.57%
	Qwen-3-30B-A3B-Instruct-2507														
256k	Dense	100.00%	100.00%	100.00%	97.00%	100.00%	99.00%	99.75%	99.25%	77.60%	26.70%	92.33%	61.00%	50.00%	84.82%
	FlexPrefill	100.00%	100.00%	100.00%	100.00%	96.00%	96.00%	98.25%	99.75%	92.60%	10.19%	92.66%	62.00%	52.00%	**84.57%**
	PBS-Attn	100.00%	99.00%	100.00%	87.00%	94.00%	94.00%	95.75%	99.25%	76.00%	18.90%	86.66%	66.00%	50.00%	82.04%
	PRISM	100.00%	100.00%	100.00%	86.00%	93.00%	94.00%	96.50%	99.50%	76.60%	26.90%	93.00%	66.00%	47.00%	82.96%
	Triceps ($\tau = 0.4$)	100.00%	100.00%	99.00%	99.00%	96.00%	95.00%	98.75%	100.00%	75.40%	8.40%	93.67%	59.00%	50.00%	82.63%
	Llama-3.1-Nemotron-8B-UltraLong-4M														
	Dense	100.00%	87.00%	100.00%	79.00%	71.00%	6.00%	77.25%	83.75%	88.40%	0.00%	75.66%	53.00%	31.00%	65.54%
	FlexPrefill	98.00%	88.00%	93.00%	54.00%	65.00%	19.00%	75.25%	84.25%	90.00%	0.60%	66.99%	41.00%	33.00%	62.16%
	PBS-Attn	99.00%	85.00%	98.00%	70.00%	66.00%	7.00%	68.50%	79.00%	91.00%	0.20%	69.96%	37.00%	32.00%	61.74%
	PRISM	100.00%	86.00%	100.00%	76.00%	65.00%	11.00%	73.00%	82.00%	90.00%	0.00%	64.00%	38.00%	30.00%	**62.69%**
	Triceps ($\tau = 0.4$)	100.00%	74.00%	96.00%	58.00%	65.00%	25.00%	74.00%	80.25%	77.80%	0.10%	65.67%	35.00%	33.00%	60.29%

Figure 3 reports the speed-up over dense attention; the table below lists the absolute time-to-first-token values it is derived from.

Table 5: Time To First Token: 64k to 256k with TP=2; Triceps uses $k_c = 0.5$, $\tau = 0.4$.

(a). LongReason

Method	64k	128k	256k
Qwen-3-30B-A3B			
Dense	6.16s	22.67s	80.06s
FlexPrefill	**4.06s**	<u>10.77s</u>	<u>28.98s</u>
PBS-Attn	4.98s	13.47s	35.19s
PRISM	<u>4.32s</u>	12.66s	33.82s
Triceps ($\tau = 0.4$)	4.47s	**9.81s**	**23.71s**
Llama-3.1-Nemotron-8B-UltraLong-4M			
Dense	5.25s	17.48s	63.79s
FlexPrefill	<u>4.54s</u>	<u>11.55s</u>	<u>35.59s</u>
PBS-Attn	4.91s	12.57s	36.04s
PRISM	**4.38s**	12.15s	37.21s
Triceps ($\tau = 0.4$)	4.62s	**9.82s**	**24.33s**

(b). RULER (niah_multikey_2)

Method	64k	128k	256k
Qwen-3-30B-A3B			
Dense	6.11s	22.57s	79.99s
FlexPrefill	**5.57s**	<u>16.51s</u>	<u>45.82s</u>
PBS-Attn	<u>5.63s</u>	16.88s	48.39s
PRISM	5.74s	17.99s	52.86s
Triceps ($\tau = 0.4$)	5.90s	**15.13s**	**40.11s**
Llama-3.1-Nemotron-8B-UltraLong-4M			
Dense	5.23s	17.43s	63.79s
FlexPrefill	**4.90s**	<u>13.24s</u>	<u>41.81s</u>
PBS-Attn	5.12s	14.55s	44.77s
PRISM	5.12s	15.09s	49.10s
Triceps ($\tau = 0.4$)	5.12s	**11.88s**	**31.13s**

B PROOFS

B.1 MATRIX-FREE ATTENTION USING DECOMPOSITION

Proof of equivalence in Equation 2, i.e. that for all $k \in \{1, \dots, d\}$

$$(\boldsymbol{H} \odot \boldsymbol{D} \odot \boldsymbol{C})\boldsymbol{V}_{:,k} = \boldsymbol{h} \odot (\boldsymbol{d} * (\boldsymbol{c} \odot \boldsymbol{V}_{:,k})),$$

where $\boldsymbol{H} = \boldsymbol{h}\boldsymbol{1}^\top$, $D_{i,j} = \boldsymbol{d}_{i-j}$, and $\boldsymbol{C} = \boldsymbol{1}\boldsymbol{c}^\top$.

Equivalently, if we allow the short-hand $\boldsymbol{v} = \boldsymbol{V}_{:,k:}$, then the proof goals is

$$\sum_{j=1}^{N} h_i d_{i-j} c_j v_j = h_i \sum_{j=-\infty}^{\infty} d_{i-j} c_j v_j,$$

which is indeed the case because by definition $d_t = 0$ for $t \leq 0$ or when $t > N$

B.2 ERROR OF THE CALIBRATED APPROXIMATION

The convolution calibration makes the approximation $\widehat{\boldsymbol{A}}$ row-stochastic, so its rows can be read as attention rows of the perturbed scores $\widehat{s}_{i,:} = s_{i,:} - e_{i,:}$.

The attention error then gets a simpler form, namely

$$\big[(\boldsymbol{A} - \widehat{\boldsymbol{A}})\boldsymbol{V}\big]_i = \mathbb{E}_{p_{i,0}}[\boldsymbol{v}] - \mathbb{E}_{p_{i,1}}[\boldsymbol{v}] = \int_0^1 \mathrm{Cov}_{p_{i,t}}(e_{i,:}, \boldsymbol{v})\, dt, \qquad p_{i,t} = \mathrm{softmax}\big(s_{i,:} - t\, e_{i,:}\big), \tag{20}$$

where $s_{i,:}$ are the exact pre-softmax scores of row i, $e_{i,:} = s_{i,:} - \widehat{s}_{i,:}$ are the residuals of the linear least-squares at row i (including the row-centring drift terms), $p_{i,0} = \mathrm{softmax}(s_{i,:})$ and $p_{i,1} = \mathrm{softmax}(\widehat{s}_{i,:})$ are the exact and the fitted attention rows, and $\boldsymbol{v}$ denotes the row of value vectors. In practice, just $\mathrm{Cov}_{p_{i,0}}(e_{i,:}, \boldsymbol{v})$ is already a good approximation.

Proof. For the path $p_t = \mathrm{softmax}(s - te)$, the derivative of the expectation is

$$\frac{d}{dt}\mathbb{E}_{p_t}[\boldsymbol{v}] = \sum_j \dot{p}_{t,j}\, \boldsymbol{v}_j, \qquad \dot{p}_{t,j} = p_{t,j}\big(-e_j + \mathbb{E}_{p_t}[e]\big).$$

Hence

$$\frac{d}{dt}\mathbb{E}_{p_t}[\boldsymbol{v}] = -\big(\mathbb{E}_{p_t}[e\boldsymbol{v}] - \mathbb{E}_{p_t}[e]\mathbb{E}_{p_t}[\boldsymbol{v}]\big) = -\mathrm{Cov}_{p_t}(e, \boldsymbol{v}),$$

and integrating from $t = 0$ to $t = 1$ yields equation 20. This is the standard covariance identity of exponential families; see e.g. Brown (1986, Ch. 2). □

The identity shows that the output error is governed by the residuals at the positions where attention is high. A score-level statistic computed on uniformly sampled scores, such as the coefficient of determination of the fit,

$$R^2 = 1 - \frac{\sum_j e_{i,j}^2}{\sum_j (s_{i,j} - \bar{s}_i)^2},$$

which measures how much of the score variance the fit explains, cannot capture it. This is why the convolution mode is gated by the measured worst-row output error E_{conv} (Equation 15).

C IMPLEMENTATION DETAILS

Algorithm 1 gives an overview of the whole method. It decomposes naturally in procedures for computing the Λ-shape as Algorithm 2, the decomposition as Algorithm 3, the statistics-based calibration as Algorithm 4, the routing measurement as Algorithm 5 and the per-head decision as Algorithm 6, the convolution-based calibration as Algorithm 7, and the three different attention kernels as Algorithm 8 for the convolution kernel, Algorithm 9 for the block-sparse kernel, and Algorithm 10 for the delegated attention kernel. Table 6 summarizes per algorithm the flop count with precise leading-order constants.

Table 6: Leading-order floating-point operation count for each subroutine. Constants follow the convention that an $m \times n$ matrix-vector product costs $2mn$ flops, a length-N real FFT costs $\sim 2.5\, N \log_2 N$ flops, a complex multiply costs 6 flops, and convolution FFTs are evaluated at the twice-padded (linear-convolution) length. The notation ρ denotes the realized block density, n_s the least-squares sample size, $|\mathbb{J}|$ the row sample size for statistics estimation, k the number of measured tail rows, B the block size, and H the number of query heads per KV group; the fit's linear-algebra terms are evaluated once per group and listed at that cost; the Total rows are per query head, with the group-shared fit terms amortised over the group's H query heads. The block-sparse mask level is resolved once in RoutingMeasurement and passed to the kernel. The overhead row collects what every head pays before the mode decision (statistics calibration, routing measurement, Λ-shape and the fit's group-shared terms); each Total adds only the path it executes.

Algorithm	**Leading-order FLOPs**	
`ComputeLambdaShape`	$4N(S+T)d$	
`FitDecomposition`	$2Nd$	(centering quantities)
	$+\,2N(S{+}T)d$	(exact sink/window logsumexps)
	$+\,2n_s d$	(sample scores at n_s pairs)
	$+\,4n_s d^2$	(normal equations)
	$+\,8n_s dH$	(cross term $\boldsymbol{A}^\top \boldsymbol{B}$ and prediction)
	$+\,\frac{8}{3}d^3$	(solve)
	$+\,4Nd$	(assemble $\boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\delta}, \boldsymbol{\lambda}$)
`CalibrateHorizontal_Convolution`	$15\, N \log_2 N$	(convolution $\boldsymbol{d} * \boldsymbol{c}$: 3 real FFTs)
	$+\,4N$	(assemble $\boldsymbol{A}'_\Lambda \mathbf{1} + \boldsymbol{g} \odot (\boldsymbol{d} * \boldsymbol{c})$, divide)
`RoutingMeasurement`	$4kNd$	(fused pass over the k rows)
	$+\,\mathcal{O}(kNd/B)$	(mask validity checks)
	$+\,\mathcal{O}(kNd)$	(conv output at the measured rows)
	$+\,\mathcal{O}((N/B)^2)$	(level and density replay)
`ComputeAttention_Convolution`	$10\, d\, N \log_2 N$	($2d{+}1$ real FFTs)
	$+\,4Nd$	(element-wise ops with $\boldsymbol{c}$ and $\boldsymbol{h}$)
`CalibrateHorizontal_Statistics`	$2Nd^2$	(row-wise variance via $\boldsymbol{K}_{\text{int}}^\top \boldsymbol{K}_{\text{int}}$)
	$+\,4\lvert\mathbb{J}\rvert Nd$	(exact $\mathbb{E}[\exp(\cdot)]$ on sampled rows)
	$+\,6N$	(per-row ν_i, h_i)
`ComputeAttention_BlockSparse`	$4\rho N^2 d$	(block-sparse kernel; level passed in)
`ComputeAttention_Delegation`	FlexPrefill terms	(default delegation target)
	or $4N^2 d$	(full attention, FA2)
Overhead, all modes (T_{ovh})	$2Nd^2 + \mathcal{O}(kNd + (N/B)^2) + \mathcal{O}\big(N(S{+}T)d + n_s d + (n_s d^2 + d^3)/H\big)$	
Total — convolution mode	$T_{\text{ovh}} + 10\, d\, N \log_2 N + 15\, N \log_2 N$	
Total — block-sparse mode	$T_{\text{ovh}} + 4\rho N^2 d$	
Total — delegation mode	T_{ovh} + FlexPrefill terms or $4N^2 d$ (FA2)	

Algorithm 1 TRICEPS TOP-LEVEL OVERVIEW (per head, during prefill)

Require: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N\times d}$; sink width S; window width T; block size B; tail-row count k; error budget τ
Ensure: $\boldsymbol{AV}$

$\boldsymbol{A}'_\Lambda \boldsymbol{V} \leftarrow$ COMPUTELAMBDASHAPE$(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, S, T)$
$(\boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\mu}, \boldsymbol{\delta}, \boldsymbol{\lambda}) \leftarrow$ FITDECOMPOSITION$(\boldsymbol{Q}, \boldsymbol{K}, S, T)$
$(\boldsymbol{h}, \boldsymbol{\nu}) \leftarrow$ CALIBRATEHORIZONTAL_STATISTICS$(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{\mu}, \boldsymbol{\delta}, \boldsymbol{\lambda})$
$(E_{\text{conv}}, \theta, E_{\text{sparse}}, M_{\text{drop}}, \rho) \leftarrow$ ROUTINGMEASUREMENT$(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{d}, \boldsymbol{c}, \boldsymbol{h}, \boldsymbol{\mu}, \boldsymbol{\delta}, \boldsymbol{\lambda}, S, T, B, k, \tau)$
$m \leftarrow$ DECIDEHEAD$(E_{\text{conv}}, M_{\text{drop}}, \rho, \tau)$
if $m =$ conv **then**
  $\boldsymbol{h} \leftarrow$ CALIBRATEHORIZONTAL_CONVOLUTION$(\boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\delta}, \boldsymbol{\lambda}, \boldsymbol{A}'_\Lambda \mathbf{1})$
  $\boldsymbol{g} \leftarrow \exp(-(\boldsymbol{\delta} + \boldsymbol{\lambda}))$
  return COMPUTEATTENTION_CONVOLUTION$(\boldsymbol{h}, \boldsymbol{g}, \boldsymbol{d}, \boldsymbol{c}, \boldsymbol{V}, \boldsymbol{A}'_\Lambda \boldsymbol{V})$
else if $m =$ sparse **then**
  return COMPUTEATTENTION_BLOCKSPARSE$(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{h}, \boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\nu}, \boldsymbol{\mu}, B, \theta)$
else
  return COMPUTEATTENTION_DELEGATION$(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$
end if

Algorithm 2 COMPUTELAMBDASHAPE

Require: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$; sink width S; window width T
Ensure: $\boldsymbol{A}'_\Lambda \boldsymbol{V}$ ▷ unnormalised band output

Compute $[\boldsymbol{A}'_\Lambda \boldsymbol{V}]_i$ exactly using only the keys/values at positions $j \in \{1, \dots, S\} \cup \{i - T + 1, \dots, i\}$
return $\boldsymbol{A}'_\Lambda \boldsymbol{V}$

Algorithm 3 FITDECOMPOSITION

Require: $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{N\times d}$; sink width S; window width T
Ensure: $\boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\mu}, \boldsymbol{\delta}, \boldsymbol{\lambda} \in \mathbb{R}^L$ ▷ $L = N - S - T$: interior only

Sample up to 10^4 index pairs $\mathbb{I} \subset \{(i,j) : S < j \le i - T,\ S + T < i \le N\}$ uniformly from the interior
Recover the pre-RoPE keys $\tilde{\boldsymbol{k}}_j \leftarrow \boldsymbol{k}_j \boldsymbol{R}_j^{-1}$ for the interior keys $j \in \{S + 1, \dots, N - T\}$
Compute row-wise centering quantities for $i \in \{S + T + 1, \dots, N\}$:
  $\overline{\boldsymbol{r}}_i \leftarrow \frac{1}{i-S-T} \sum_{j=S+1}^{i-T} \boldsymbol{r}_{j-i}$
  $\overline{\tilde{\boldsymbol{k}}}_i \leftarrow \frac{1}{i-S-T} \sum_{j=S+1}^{i-T} \tilde{\boldsymbol{k}}_j$
  $\mu_i \leftarrow \frac{1}{i-S-T} \sum_{j=S+1}^{i-T} \boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d}$
Solve the ridge-regularized least-squares problem:
  $(\boldsymbol{\alpha}, \boldsymbol{\kappa}) \leftarrow \arg\min_{\boldsymbol{\alpha},\boldsymbol{\kappa}} \sum_{(i,j)\in\mathbb{I}} \left((\boldsymbol{r}_{j-i} - \overline{\boldsymbol{r}}_i)\boldsymbol{\alpha} + (\tilde{\boldsymbol{k}}_j - \overline{\tilde{\boldsymbol{k}}}_i)\boldsymbol{\kappa} - (\boldsymbol{q}_i \boldsymbol{k}_j^\top / \sqrt{d} - \mu_i) \right)^2 + \gamma(\|\boldsymbol{\alpha}\|^2 + \|\boldsymbol{\kappa}\|^2)$,
  where $\gamma = \frac{1}{4}\varepsilon_r^2 \sum_{(i,j)\in\mathbb{I}} \left(\|\boldsymbol{r}_{j-i} - \overline{\boldsymbol{r}}_i\|^2 + \|\tilde{\boldsymbol{k}}_j - \overline{\tilde{\boldsymbol{k}}}_i\|^2 \right)$, $\varepsilon_r = 2^{-7}$
for $t \in \{T, \dots, N - S - 1\}$ **do**
  $d_t \leftarrow \exp\left((\boldsymbol{r}_{-t} - \overline{\boldsymbol{r}}_N)\boldsymbol{\alpha}\right)$ ▷ slash component, indexed by $t = i - j$
end for
for $j \in \{S + 1, \dots, N - T\}$ **do**
  $c_j \leftarrow \exp\left((\tilde{\boldsymbol{k}}_j - \overline{\tilde{\boldsymbol{k}}}_N)\boldsymbol{\kappa}\right)$ ▷ vertical component
end for
for $i \in \{S + T + 1, \dots, N\}$ **do** ▷ per-row corrections, defined over interior rows only
  $\delta_i \leftarrow (\overline{\boldsymbol{r}}_i - \overline{\boldsymbol{r}}_N)\boldsymbol{\alpha}$
  $\lambda_i \leftarrow (\overline{\tilde{\boldsymbol{k}}}_i - \overline{\tilde{\boldsymbol{k}}}_N)\boldsymbol{\kappa}$
end for
return $(\boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\mu}, \boldsymbol{\delta}, \boldsymbol{\lambda})$

Algorithm 4 CALIBRATEHORIZONTAL_STATISTICS

Require: $\boldsymbol{Q}, \boldsymbol{K} \in \mathbb{R}^{N \times d}$; $\boldsymbol{\mu}, \boldsymbol{\delta}, \boldsymbol{\lambda} \in \mathbb{R}^N$ (defined for interior rows)
Ensure: $\boldsymbol{h}, \boldsymbol{\nu} \in \mathbb{R}^N$ (defined for interior rows)
Compute $\sigma_i^2 \leftarrow \mathrm{Var}(\boldsymbol{q}_i \boldsymbol{K}^\top / \sqrt{d})$ for $i \in \{S+T+1, \ldots, N\}$
Sample ~ 10 rows $\mathbb{J} \subset \{S+T+1, \ldots, N\}$
for $i \in \mathbb{J}$ **do**
    Compute $\mathbb{E}[\exp(\boldsymbol{q}_i \boldsymbol{k}^\top / \sqrt{d} - \mu_i)]$ exactly over $j \in \{S+1, \ldots, i-T\}$
end for
Fit constant c such that $\mathbb{E}[\exp(\boldsymbol{q}_i \boldsymbol{k}^\top / \sqrt{d} - \mu_i)] \approx \exp(c\sigma_i^2)$ over $\mathbb{J}$
for $i \in \{S+T+1, \ldots, N\}$ **do**
    $\mathrm{lse}_{\mathrm{mid}} \leftarrow \log(i - S - T) + c\sigma_i^2$
    $\nu_i \leftarrow \log\big(\exp(\mathrm{lse}_{\mathrm{sink}}) + \exp(\mathrm{lse}_{\mathrm{mid}}) + \exp(\mathrm{lse}_{\mathrm{diag}})\big)$ ▷ full-row softmax denominator
    $h_i \leftarrow \exp(-\delta_i - \lambda_i - \nu_i)$
end for
return $(\boldsymbol{h}, \boldsymbol{\nu})$

Algorithm 5 ROUTINGMEASUREMENT

Require: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$; $\boldsymbol{d}, \boldsymbol{c}, \boldsymbol{h}, \boldsymbol{\mu}, \boldsymbol{\delta}, \boldsymbol{\lambda}$; S, T; block size B; tail count k; budget τ
Ensure: $E_{\mathrm{conv}}, \theta, M_{\mathrm{drop}}, \rho$
$\mathcal{R} \leftarrow$ the k deepest interior rows ▷ 64 last rows
$d(\tau) \leftarrow 1 - \exp(-k_c \tau)$ ▷ dropped-mass budget
for each head h **do**
    $E_{\mathrm{conv}}[h] \leftarrow \max_{i \in \mathcal{R}} \|\boldsymbol{O}_i^{\mathrm{conv}} - \boldsymbol{O}_i^{\mathrm{dense}}\|_2 / \|\boldsymbol{O}_i^{\mathrm{dense}}\|_2$ ▷ drift-corrected conv
    $n_i^* \leftarrow$ smallest score-ordered count with true (forced + prefix) mass $\geq 1 - d(\tau)$
    $p \leftarrow \mathrm{mean}_{i \in \mathcal{R}}\, b_i / \lceil N/B \rceil$ ▷ $b_i = n_i^* + |\mathrm{forced}_i|$: blocks kept by row i
    $\theta[h] \leftarrow (1-p)$-quantile of $\widetilde{c}_b + \widetilde{h}_q + \widetilde{d}_{b-q} + \log(q+1)$ over the causal triangle
    $M_{\mathrm{drop}}[h] \leftarrow \mathrm{mean}_{i \in \mathcal{R}}$ dropped mass of the θ-mask ▷ tail-row mean discarded attention mass
    $E_{\mathrm{sparse}}[h] \leftarrow \max_{i \in \mathcal{R}}$ relative ℓ_2 error of the θ-mask
    $\rho[h] \leftarrow$ mean deployed density over the full mask
end for
return $(E_{\mathrm{conv}}, \theta, E_{\mathrm{sparse}}, M_{\mathrm{drop}}, \rho)$

Algorithm 6 DECIDEHEAD

Require: $E_{\mathrm{conv}}, M_{\mathrm{drop}}, \rho, \tau$; cost model $\mathcal{C}$
Ensure: mode m
if $E_{\mathrm{conv}} \leq \tau$ **then**
    return conv
end if
$d(\tau) \leftarrow 1 - \exp(-k_c \tau)$
$\mathcal{V} \leftarrow \{\}$
if $M_{\mathrm{drop}} \leq d(\tau)$ and $\rho \leq \rho_0$ **then**
    $\mathcal{V}[\mathrm{sparse}] \leftarrow \min\big(\mathcal{C}_{\mathrm{sparse}}(\rho), \mathcal{C}_{\mathrm{dense}}\big)$
end if
$\mathcal{V}[\mathrm{delegation}] \leftarrow \mathcal{C}_{\mathrm{deleg}}$
return $\arg\min_{m \in \mathcal{V}} \mathcal{V}[m]$

Algorithm 7 CALIBRATEHORIZONTAL_CONVOLUTION

Require: $\boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\delta}, \boldsymbol{\lambda}$, $\boldsymbol{A}'_\Lambda \mathbf{1} \in \mathbb{R}^N$ (unnormalised Λ-shape row sums, i.e. the exponentiated sink/window logsumexps)
Ensure: $\boldsymbol{h} \in \mathbb{R}^N$ (row-normalisation multiplier)
$\boldsymbol{g} \leftarrow \exp(-(\boldsymbol{\delta} + \boldsymbol{\lambda}))$ ▷ row-centring drift of the fit
$\boldsymbol{h} \leftarrow \mathbf{1} \,/\, \big(\boldsymbol{A}'_\Lambda \mathbf{1} + \boldsymbol{g} \odot (\boldsymbol{d} * \boldsymbol{c})\big)$ ▷ exact row mass
return $\boldsymbol{h}$

Algorithm 8 COMPUTEATTENTION_CONVOLUTION

Require: $\boldsymbol{h}, \boldsymbol{g}, \boldsymbol{d}, \boldsymbol{c}, \boldsymbol{V}$, $\boldsymbol{A}'_\Lambda \boldsymbol{V}$ (unnormalised bands)
Ensure: $\boldsymbol{AV}$
1: $\boldsymbol{O}_{\text{int}} \leftarrow \boldsymbol{g} \odot \big(\boldsymbol{d} * (\boldsymbol{c} \odot \boldsymbol{V})\big)$ ▷ drift-corrected interior; convolution via FFT, $\mathcal{O}(N \log N)$
2: **return** $\boldsymbol{h} \odot \big(\boldsymbol{A}'_\Lambda \boldsymbol{V} + \boldsymbol{O}_{\text{int}}\big)$ ▷ bands and interior share the row multiplier $\boldsymbol{h}$

Algorithm 9 COMPUTEATTENTION_BLOCKSPARSE

Require: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$; $\boldsymbol{h}, \boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\nu}, \boldsymbol{\mu}$; block size B; mask level θ
Ensure: $\boldsymbol{AV}$
Step 1: Block-level approximation in log-domain
1: $\widetilde{h}_m \leftarrow \max_{i \in \text{block}(m)} \log h_i$ for each query block m
2: $\widetilde{c}_n \leftarrow \max_{j \in \text{block}(n)} \log c_j$ for each key block n
3: $\widetilde{d}_t \leftarrow$ boxcar average over $\lceil \max(B_q, B_{kv}) / \min(B_q, B_{kv}) \rceil + 1$ adjacent offset blocks of $\max_{u \in \text{block}(t)} \log d_u$ ▷ diagonal only; two blocks when $B_q = B_{kv}$
4: $\widetilde{\boldsymbol{A}}_{\text{block}}[m, n] \leftarrow \widetilde{h}_m + \widetilde{d}_{m-n} + \widetilde{c}_n + \log(m+1)$ for $1 \le n \le m \le \lceil N/B \rceil$
Step 2: Mask level from the routing measurement
5: $\theta \leftarrow$ the level resolved in ROUTINGMEASUREMENT ▷ $(1-p)$-quantile of the executed statistic
Step 3: Mask construction and kernel call
6: $\mathcal{M} \leftarrow \{(m, n) : \widetilde{\boldsymbol{A}}_{\text{block}}[m, n] \ge \theta\} \cup \{(m, n) : \text{block}(m, n) \text{ overlaps the } \Lambda\text{-shape}\}$
7: **return** BLOCKSPARSEATTENTIONKERNEL$(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \mathcal{M})$

Algorithm 10 COMPUTEATTENTION_DELEGATION

Require: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$
Ensure: $\boldsymbol{AV}$
1: **return** result of FlexPrefill (Lai et al., 2025) on $(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$
▷ default delegation target; FA2 when the dense delegation flag is enabled