

SPARSER BLOCK-SPARSE ATTENTION VIA TOKEN PERMUTATION

Xinghao Wang¹, Pengyu Wang¹, Dong Zhang¹, Chenkun Tan¹, Shaojun Zhou¹,

Zhaoxiang Liu², Shiguo Lian², Fangxu Liu³, Kai Song³, Xipeng Qiu^{1,4}

¹Fudan University, ²China Unicom, ³ByteDance, ⁴Shanghai Innovation Institute

ABSTRACT

Scaling the context length of large language models (LLMs) offers significant benefits but is computationally expensive. This expense stems primarily from the self-attention mechanism, whose $O(N^2)$ complexity with respect to sequence length presents a major bottleneck for both memory and latency. Fortunately, the attention matrix is often sparse, particularly for long sequences, suggesting an opportunity for optimization. Block-sparse attention has emerged as a promising solution that partitions sequences into blocks and skips computation for a subset of these blocks. However, the effectiveness of this method is highly dependent on the underlying attention patterns, which can lead to sub-optimal block-level sparsity. For instance, important key tokens for queries within a single block may be scattered across numerous other blocks, leading to computational redundancy. In this work, we propose Permuted Block-Sparse Attention (**PBS-Attn**), a plug-and-play method that leverages the permutation properties of attention to increase block-level sparsity and enhance the computational efficiency of LLM prefilling. We conduct comprehensive experiments on challenging real-world long-context datasets, demonstrating that PBS-Attn consistently outperforms existing block-sparse attention methods in model accuracy and closely matches the full attention baseline. Powered by our custom permuted-FlashAttention kernels, PBS-Attn achieves an end-to-end speedup of up to $2.75\times$ in long-context prefilling, confirming its practical viability. Code available at <https://github.com/xinghaow99/pbs-attn>.

1 INTRODUCTION

Modern Large Language Models (LLMs) have demonstrated remarkable proficiency in handling long-context tasks (OpenAI, 2025; Gemini Team, Google, 2025; Anthropic, 2025), a capability fueled by advancements in infrastructure (Liu et al., 2023; Jin et al., 2024), training methodologies (Yang et al., 2025a), and novel positional embedding schemes (Su et al., 2023; Press et al., 2022; Peng et al., 2023). This progress enables models to process context windows spanning thousands or even millions of tokens, unlocking novel applications such as analyzing entire codebases, summarizing lengthy legal documents, and interpreting long-form video content.

However, this extended capability is constrained by prohibitive memory and computational overheads. This bottleneck primarily stems from the self-attention mechanism within the Transformer architecture (Vaswani et al., 2023). The necessity for each token to attend to all other tokens results in a computational complexity that scales quadratically with the input sequence length, posing a fundamental challenge to scalable and accessible long-context processing.

To address this challenge, researchers have proposed solutions from multiple perspectives. Architecturally, some approaches replace the standard quadratic attention with sub-quadratic alternatives, such as linear transformers (Katharopoulos et al., 2020; Yang et al., 2025d). Others substitute the attention mechanism entirely with alternatives like State Space Models (SSMs), which operate recurrently to process extremely long sequences with high efficiency (Gu & Dao, 2024; Dao & Gu, 2024; Yang et al., 2025c). Concurrently, hardware-aware optimizations, exemplified by FlashAttention (Dao et al., 2022), reduce memory overhead by tiling the sequence into blocks and performing an online softmax computation. This method avoids the materialization of the full attention ma-

trix, thereby alleviating the memory overhead and efficiency constraints imposed by I/O limitations. Building directly upon this tiled approach, block-sparse attention further reduces computation by skipping the computation for certain blocks using a pre-computed sparse block mask (Dao et al., 2022; Jiang et al., 2024; Lai et al., 2025; Xu et al., 2025; Zhang et al., 2025; Gao et al., 2025). This technique leverages the inherent sparsity of attention matrices, wherein most of the attention mass for a given query is concentrated on a small subset of key tokens. This property, particularly prominent in long sequences, allows for a drastic reduction in computation without significantly compromising performance. While this block-level approach maximizes parallel efficiency, its rigidity can lead to a sub-optimal sparsity pattern. This issue arises when the key tokens relevant to queries within a single block are widely scattered, collectively spanning an unnecessarily large number of key blocks and thereby forcing redundant computation.

Fortunately, the same token-wise computation that leads to quadratic complexity also presents an opportunity to mitigate it. The attention mechanism is permutation-invariant, meaning we can reorder the query and key sequences to achieve a more favorable block-sparse structure and further improve block sparsity. Leveraging this insight, we propose **Permuted Block-Sparse Attention (PBS-Attn)**, a plug-and-play strategy that reorganizes query and key sequences to accelerate LLM prefilling. To accommodate causal attention for LLMs, we introduce a novel segmented permutation strategy that preserves inter-segment causality while applying intra-segment permutation. Extensive experiments demonstrate that PBS-Attn increases block-level sparsity, yielding significant efficiency gains with minimal degradation in model performance. Specifically, powered by our custom permuted-FlashAttention kernels, PBS-Attn achieves an end-to-end speedup of up to $2.75 \times$ in LLM prefilling, while maintaining performance close to the full attention baseline on real-world datasets like LongBench (Bai et al., 2024) and LongBenchv2 (Bai et al., 2025).

2 PRELIMINARIES

Scaled Dot-Product Attention As the cornerstone of modern large language models, the attention mechanism facilitates a dynamic synthesis of information by calculating a weighted aggregation of value (\mathbf{V}) vectors. These weights, or attention scores, are determined by the dot-product similarity between a given token’s query (\mathbf{Q}) vector and the key (\mathbf{K}) vectors of all other tokens in the sequence. This process allows the model to directly assess the relevance of every token relative to every other, enabling the effective capture of long-range dependencies, but at a cost of quadratic complexity over the sequence length. Formally, the attention mechanism is given by:

$$\mathbf{A} = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) \quad (1)$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A}\mathbf{V} \quad (2)$$

where d is the head dimension for multi-head attention and \mathbf{A} is the attention matrix.

FlashAttention FlashAttention (Dao et al., 2022) employs a tiled approach that partitions the input sequence into blocks and performs an online softmax computation. This strategy circumvents the materialization of the full attention matrix \mathbf{A} , which significantly reduces memory overhead and improves efficiency for I/O-bound operations on GPUs.

Formally, let the input query, key and value matrices be $\mathbf{Q} \in \mathbb{R}^{N \times d}$, $\mathbf{K} \in \mathbb{R}^{M \times d}$, and $\mathbf{V} \in \mathbb{R}^{M \times d}$ and divide them into $T_r = \lceil \frac{N}{B} \rceil$ and $T_c = \lceil \frac{M}{B} \rceil$ blocks with block size B (we use the same block size for \mathbf{Q} and \mathbf{K}/\mathbf{V} for simple terminology), $\mathbf{Q} = [\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}]$, $\mathbf{K} = [\mathbf{K}_1, \dots, \mathbf{K}_{T_c}]$, and $\mathbf{V} = [\mathbf{V}_1, \dots, \mathbf{V}_{T_c}]$. For query block \mathbf{Q}_i , the computation for the corresponding output block \mathbf{O}_i is defined by a system of recursive equations over the key/value blocks $j = 1, \dots, T_c$. The state at step j is the triplet $(\mathbf{O}_i^{(j)}, \mathbf{m}_i^{(j)}, \mathbf{l}_i^{(j)})$. The state is initialized at $j = 0$ with $\mathbf{O}_i^{(0)} = \mathbf{0}$, $\mathbf{m}_i^{(0)} = -\infty$, and $\mathbf{l}_i^{(0)} = \mathbf{0}$. For each step $j = 1, \dots, T_c$, given the intermediate scores $S_{ij} = \frac{\mathbf{Q}_i \mathbf{K}_j^T}{\sqrt{d}}$ and local maximum $\mathbf{m}'_{ij} = \text{row_max}(\mathbf{S}_{ij})$, the state is updated from $j - 1$ to j :

$$\mathbf{m}_i^{(j)} = \max(\mathbf{m}_i^{(j-1)}, \mathbf{m}'_{ij}) \quad (3)$$

$$\mathbf{l}_i^{(j)} = \mathbf{l}_i^{(j-1)} e^{\mathbf{m}_i^{(j-1)} - \mathbf{m}_i^{(j)}} + \text{row_sum}(\exp(\mathbf{S}_{ij} - \mathbf{m}_i^{(j)})) \quad (4)$$

$$\mathbf{O}_i^{(j)} = \mathbf{O}_i^{(j-1)} e^{\mathbf{m}_i^{(j-1)} - \mathbf{m}_i^{(j)}} + \exp(\mathbf{S}_{ij} - \mathbf{m}_i^{(j)}) \mathbf{V}_j \quad (5)$$

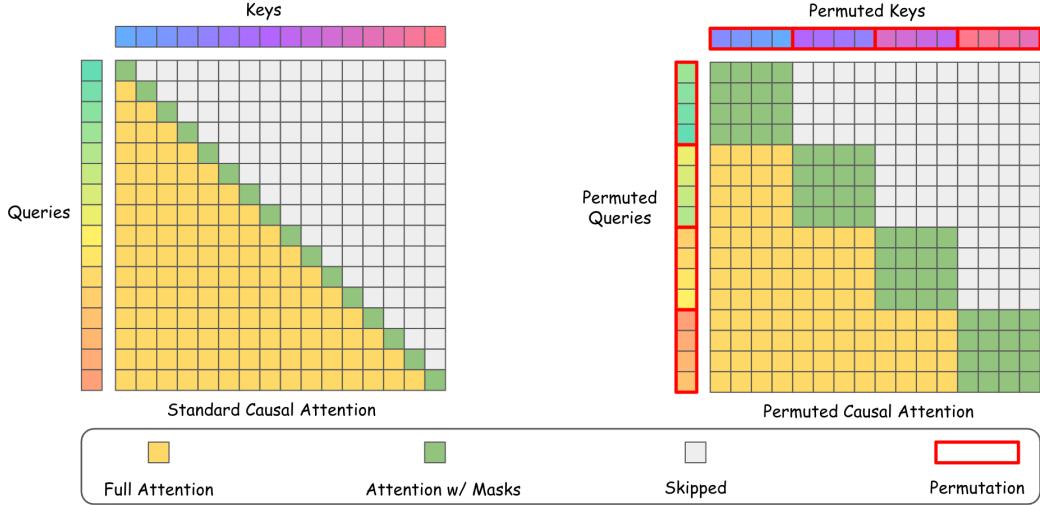


Figure 1: Illustration of causal attention without (**Left**) and with (**Right**) segmented permutation with $B = 1, S = 4$. Segmented permutation enhances block-level sparsity via intra-segment permutation while preserving inter-segment causality. By restricting computation of blocks within on-diagonal segments (green blocks), we can safely skip inter-segment blocks (yellow blocks) for block-sparse attention.

After the final step, the output is normalized as $\mathbf{O}_i = \text{diag}\left((\mathbf{l}_i^{(T_c)})^{-1}\right) \mathbf{O}_i^{(T_c)}$.

Block-Sparse Attention Building upon the tiled computation of FlashAttention, block-sparse attention introduces a further layer of optimization by selectively pruning block-wise interactions. This is achieved using a predefined sparse block mask, $\mathbf{M} \in \{0, 1\}^{T_r \times T_c}$. For any given query block \mathbf{Q}_i , the attention computation is only performed against key-value blocks \mathbf{K}_j and \mathbf{V}_j where the corresponding mask entry $\mathbf{M}_{ij} = 1$.

If $\mathbf{M}_{ij} = 0$, the calculation of the score matrix \mathbf{S}_{ij} and the subsequent state update steps are entirely bypassed. Consequently, the state remains unchanged from the previous iteration; that is, $(\mathbf{O}_i^{(j)}, \mathbf{m}_i^{(j)}, \mathbf{l}_i^{(j)}) = (\mathbf{O}_i^{(j-1)}, \mathbf{m}_i^{(j-1)}, \mathbf{l}_i^{(j-1)})$.

3 PERMUTED BLOCK-SPARSE ATTENTION

3.1 PERMUTATION PROPERTIES OF ATTENTION

The attention mechanism exhibits specific symmetries with respect to permutations of its inputs, which we formalize in the following lemmas.

Lemma 3.1 (Key-Value Pair Permutation Invariance). *The attention mechanism is invariant to the order of the source sequence, provided that the key-value pairings are maintained.*

Formally, let $\mathbf{P}_\pi \in \{0, 1\}^{M \times M}$ be a permutation matrix that reorders the rows of a matrix according to a permutation π on the index set $\{1, \dots, M\}$. The following identity holds:

$$\text{Attention}(\mathbf{Q}, \mathbf{P}_\pi \mathbf{K}, \mathbf{P}_\pi \mathbf{V}) = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad (6)$$

Lemma 3.2 (Query Permutation Equivariance). *The attention mechanism is equivariant with respect to permutations of the query sequence.*

Formally, let $\mathbf{P}_\sigma \in \{0, 1\}^{N \times N}$ be a permutation matrix that reorders the rows of a matrix according to a permutation σ on the index set $\{1, \dots, N\}$. The following relationship holds:

$$\text{Attention}(\mathbf{P}_\sigma \mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{P}_\sigma \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad (7)$$

The proofs of Lemma 3.1 and 3.2 are provided in Appendix A.1 and A.2, respectively.

Combining these properties, we arrive at a general theorem for attention under simultaneous input permutations. A detailed proof is provided in Appendix A.3.

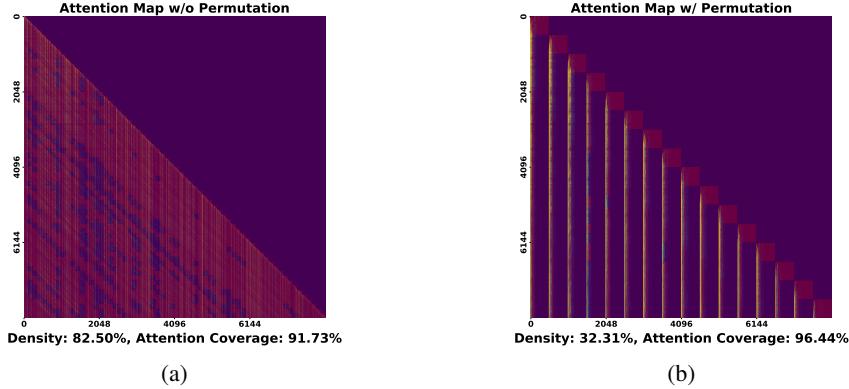


Figure 2: Comparison of attention maps for Llama-3.1-8B (layer 0, head 30) on an 8K LongBench example, showing the pattern without (a) and with (b) segmented permutation. The red overlay indicates blocks selected for block-sparse attention, and the attention coverage is calculated as the total attention scores covered by the selected blocks. More visualizations are provided in Appendix D.

Theorem 3.3 (Attention Permutation Invariance under Inverse Transformation). *If the queries are permuted by P_σ and the key-value pairs are permuted by P_π , the resulting output is a permuted version of the original output. Applying the inverse of the query permutation recovers the original, unpermuted output. Formally:*

$$P_\sigma^T \text{Attention}(P_\sigma Q, P_\pi K, P_\pi V) = \text{Attention}(Q, K, V) \quad (8)$$

Theorem 3.3 establishes that the query matrix Q and key matrix K can be permuted by P_σ and P_π respectively, provided that P_π is also applied to the value matrix V and P_σ^T to the output O' . This property enables the rearrangement of the attention matrix A , without affecting the attention output.

3.2 SEGMENTED PERMUTATION FOR CAUSAL ATTENTION

Motivated by Theorem 3.3, we explore whether its permutation properties can be leveraged to re-structure the attention matrix. This rearrangement promises higher block sparsity and computational savings, particularly during the compute-bound prefill stage of inference. The primary objective is to **co-locate the salient key tokens corresponding to queries from the same computational block, thereby enhancing block-level sparsity**.

However, a critical challenge remains: maintaining causality post-permutation. Specifically, LLMs are trained with causal attention, which restricts queries to attending only to keys in preceding positions, resulting in a lower-triangular attention matrix, A . During prefilling, blocks above the main diagonal are computationally redundant and can be skipped; consequently, the original block density for causal attention is $\frac{T_c+1}{2T_c}$. A naive application of a global permutation to the query and key sequences would dismantle this vital causal structure. Such a permutation could scatter dependencies across the entire matrix, potentially transforming the sparse, lower-triangular structure into a fully dense one (i.e., a block density of 1).

To address this challenge, we propose a segmented permutation strategy that preserves *inter-segment causality* while applying *intra-segment permutation*, illustrated in Figure 1. Formally, we partition the initial $\lfloor N/S \rfloor \cdot S$ tokens of the input sequences Q, K, V into $G = \lfloor N/S \rfloor$ non-overlapping, contiguous segments of size S . The remaining $N \pmod S$ tokens are left unpermuted.

Let $Q_i, K_i, V_i \in \mathbb{R}^{S \times d}$ denote the i -th segment for $i \in \{1, \dots, G\}$. For each segment i , we introduce local permutations, σ_i for queries and π_i for keys, that reorder tokens within that segment. The global permutation operators, P_σ and P_π , are then constructed as block-diagonal matrices from

these respective local permutations. For the key permutation matrix \mathbf{P}_π :

$$\mathbf{P}_\pi = \text{diag}(\mathbf{P}_{\pi_1}, \dots, \mathbf{P}_{\pi_G}, \mathbf{I}_{N \pmod S}) = \begin{pmatrix} \mathbf{P}_{\pi_1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_{\pi_2} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{P}_{\pi_G} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{I}_{N \pmod S} \end{pmatrix} \quad (9)$$

Here, each $\mathbf{P}_{\pi_i} \in \{0, 1\}^{S \times S}$ is the permutation matrix for the local key permutation π_i , and $\mathbf{I}_{N \pmod S}$ is the identity matrix corresponding to the last incomplete segment. The query permutation matrix \mathbf{P}_σ is constructed analogously from its own set of local permutations, $\{\sigma_i\}_{i=1}^G$.

3.3 QUERY-AWARE KEY PERMUTATION

The inherent sparsity of the attention mechanism implies that for any given query, a small subset of key tokens accounts for most of the attention mass. A prominent pattern within this distribution is that certain keys are consistently important across all queries, a phenomenon widely recognized in the literature as "Vertical Lines" (Jiang et al., 2024; Lai et al., 2025; Xu et al., 2025). To maintain model accuracy, block-sparse attention must encompass all "Vertical Lines." This constraint, however, can severely diminish block sparsity when these critical tokens are scattered throughout the sequence, spanning a large number of blocks, as shown in Figure 2a.

To leverage this sparse pattern, we introduce a query-aware key permutation strategy. This method implements the permutation as an efficient, segment-wise sorting process. Within each segment, keys are sorted based on their estimated average attention scores using the last block of queries. Concretely, we first compute a global importance score vector $\mathbf{s} \in \mathbb{R}^N$ for all keys in the sequence using the last block of queries, $\mathbf{Q}_{\text{last_block}}$:

$$\mathbf{s} = \text{mean}_{\text{rows}} \left(\text{softmax} \left(\frac{\mathbf{Q}_{\text{last_block}} \mathbf{K}^T}{\sqrt{d}} \right) \right) \quad (10)$$

The local permutation π_i for each segment i is then obtained by sorting the keys within that segment based on \mathbf{s} in descending order:

$$\pi_i = \text{argsort}(-\mathbf{s}_{[(i-1)S+1:iS]}) \quad (11)$$

As shown in Figure 2b, this permutation strategy can effectively cluster the "Vertical Lines" thereby significantly improving block-level sparsity while maintaining attention coverage.

3.4 PERMUTED BLOCK-SPARSE ATTENTION

The proposed permuted block-sparse attention (**PBS-Attn**) mechanism is detailed in Algorithm 1. The process commences by permuting the query, key, and value matrices. Subsequently, a block-sparse mask, denoted as \mathbf{M} , is derived based on the permuted queries and keys. This mask, \mathbf{M} , governs the tiled attention computation by dictating which block-wise operations can be pruned. Finally, an inverse permutation is applied to restore the original ordering of the output, established by Theorem 3.3. For the block selection algorithm, we use a simple strategy that utilizes mean pooling and block-wise attention to estimate the importance of each key block for each query block for the main method, where we detail in Appendix B.1. Crucially, we demonstrate that the sparsity improvements conferred by permutation are agnostic to the specific block selection algorithm, where we can combine the permutation with existing block selection algorithms to further improve block sparsity, as detailed in Appendix B.2.

4 EXPERIMENTS

4.1 SETTINGS

Models & Datasets We employ two state-of-the-art long-context LLMs, claiming support for available context lengths above 128K tokens: **Llama-3.1-8B(128K)** (Grattafiori et al., 2024) and

Algorithm 1 Permuted Block-Sparse Attention

Require: $Q, K, V \in \mathbb{R}^{N \times d}$, permutation matrices $P_\sigma, P_\pi \in \{0, 1\}^{N \times N}$, segment size S , block size B

Ensure: Permuted attention output $O \in \mathbb{R}^{N \times d}$

$\mathbf{Q}' \leftarrow P_\sigma \mathbf{Q}, \mathbf{K}' \leftarrow P_\pi \mathbf{K}, \mathbf{V}' \leftarrow P_\pi \mathbf{V}$ ▷ Apply permutation

Divide Q' into $T_r = \lceil \frac{N}{B} \rceil$ blocks Q'_1, \dots, Q'_{T_r} ; divide K', V' into $T_c = \lceil \frac{N}{B} \rceil$ blocks K'_1, \dots, K'_{T_c} and V'_1, \dots, V'_{T_c} ;

$M \leftarrow \text{BLOCK_SELECTION}(Q', K', B, S)$ ▷ Select blocks, see Appendix B

Initialize $O' \leftarrow 0$;

for $i = 1$ to T_r **do**

 Load Q'_i to SRAM; Initialize $O_i^{(0)} \leftarrow 0, m_i^{(0)} \leftarrow -\infty, l_i^{(0)} \leftarrow 0$;

for $j = 1$ to T_c **do**

if $M_{i,j} = 1$ **then** ▷ Compute attention only for selected blocks

 Load K'_j, V'_j to SRAM;

 Compute $S'_{ij} = Q'_i K'_j^T / \sqrt{d}, m_i^{(j)} = \max(m_i^{(j-1)}, \text{row_max}(S'_{ij}))$;

 Compute $l_i^{(j)} = l_i^{(j-1)} e^{m_i^{(j-1)} - m_i^{(j)}} + \text{row_sum}(\exp(S'_{ij} - m_i^{(j)}))$;

 Compute $O_i^{(j)} = O_i^{(j-1)} e^{m_i^{(j-1)} - m_i^{(j)}} + \exp(S'_{ij} - m_i^{(j)}) V'_j$;

else ▷ Skip computation

$O_i^{(j)} \leftarrow O_i^{(j-1)}, m_i^{(j)} \leftarrow m_i^{(j-1)}, l_i^{(j)} \leftarrow l_i^{(j-1)}$;

end if

end for

$O'_i \leftarrow \text{diag}((l_i^{(T_c)})^{-1}) O_i^{(T_c)}$; Write O'_i back to its rows in O' ;

end for

$O \leftarrow P_\sigma^T O'$ ▷ Reverse permutation

return O

Qwen-2.5-7B-1M(1M) (Yang et al., 2025a). We evaluate the sparse attention methods on two challenging real-world long-context datasets to validate their effectiveness in real-world scenarios: **LongBench**(Bai et al., 2024) and **LongBenchv2**(Bai et al., 2025). LongBench is a collection of 21 long-context understanding tasks in 6 categories with mostly real-world data, with the average length of most tasks ranging from 5K to 15K. LongBenchv2 further scales the context length, ranging from 8K to 2M, covering various realistic scenarios.

Baselines We evaluate PBS-Attn alongside a set of strong baselines to validate its effectiveness. (1) **Full Attention**: The standard attention mechanism that computes the full attention matrix as the oracle method. Specifically, we use the FlashAttention (Dao et al., 2022) implementation. (2) **Minference** (Jiang et al., 2024): A sparse attention method that performs offline attention pattern search, we utilize the official configuration for attention pattern setting. (3) **FlexPrefill** (Lai et al., 2025): A block selection method for block-sparse attention that performs block selection based on the input and selects the attention pattern on the fly. We use $\gamma = 0.95, \tau = 0.1$ as reported in the original paper. (4) **XAttention** (Xu et al., 2025): A block selection method for block-sparse attention that selects blocks based on an antidiagonal scoring of blocks. We use threshold = 0.9, stride = 8 as reported in the original paper. (5) **MeanPooling**: This method uses a mean pooling strategy on the unpermuted queries and keys to select blocks, which is the same selection method for PBS-Attn(detailed in B.1). Our experiments shows that MeanPooling can serve as a strong baseline when the first and the most recent key blocks are forcibly selected for each query block, due to the attention sink phenomenon (Xiao et al., 2024). We use a selection threshold of 0.9 for MeanPooling.

Implementation Details For PBS-Attn, we use a block size of $B = 128$ and a segment size of $S = 256$. The block selection threshold is set to 0.9 through all experiments. We implement a custom permuted-FlashAttention kernel in Triton (Tillet et al., 2019) for efficient inference of PBS-Attn. For model inference, we replace the prefilling process with PBS-Attn or baseline methods, while keeping the decoding process as in the original attention implementation. The experiments are conducted in a computing environment with NVIDIA H100 80GB GPUs.

Table 1: Performance comparison of various sparse attention methods on LongBench. **Bold** and underlined scores indicate the best and second-best performing methods in each category, respectively, with the exception of the full attention baseline.

Method	Single-Doc QA	Multi-Doc QA	Summarization	Few-shot Learning	Code	Synthetic	Avg.
<i>Llama-3.1-8B</i>							
Full	48.80	41.80	17.79	29.73	24.77	66.82	38.28
Minference	47.21	<u>40.93</u>	17.72	29.36	24.77	<u>62.36</u>	37.06
FlexPrefill	47.03	38.57	17.78	30.38	24.88	24.71	30.56
XAttention	48.26	40.23	17.86	31.35	26.19	54.64	36.42
MeanPooling	46.61	40.66	<u>17.85</u>	<u>30.64</u>	<u>26.10</u>	58.14	36.67
PBS-Attn	<u>48.00</u>	42.09	17.72	28.36	24.25	63.80	37.37
<i>Qwen-2.5-7B-1M</i>							
Full	44.21	42.97	16.01	47.48	3.91	67.50	37.01
Minference	42.82	<u>41.76</u>	16.01	46.41	3.80	66.50	36.21
FlexPrefill	38.44	37.51	15.87	46.12	6.46	26.67	28.51
XAttention	43.82	42.21	<u>16.07</u>	<u>48.30</u>	3.83	63.33	<u>36.26</u>
MeanPooling	39.39	40.96	15.95	49.07	4.80	40.83	31.83
PBS-Attn	<u>43.01</u>	41.40	16.12	47.36	4.00	<u>66.33</u>	36.37

4.2 MAIN RESULTS

LongBench Table 1 presents a performance comparison of various sparse attention methods on the LongBench benchmark, evaluated using the Llama-3.1-8B and Qwen-2.5-7B-1M models. As the results indicate, the unpermuted MeanPooling method already establishes a strong baseline. Crucially, by incorporating our proposed permutation strategy, PBS-Attn significantly improves performance, surpassing other block-sparse attention methods and closely approaching the performance of the oracle full-attention baseline. PBS-Attn consistently achieves the best overall performance across both models, demonstrating its effectiveness and robustness.

LongBenchv2 To rigorously evaluate the effectiveness of PBS-Attn in extreme long-context scenarios, we conducted experiments on the more challenging LongBenchv2 benchmark. The results, presented in Table 2, reveal that PBS-Attn exhibits minimal performance degradation compared to the full attention baseline while consistently surpassing other block-sparse attention methods. Notably, PBS-Attn consistently outperforms the unpermuted MeanPooling baseline. This advantage is particularly pronounced for the Qwen-2.5-7B-1M model, where permutation brings a remarkable relative improvement of 31% in overall performance.

Table 2: Performance comparison of various sparse attention methods on LongBenchv2. **Bold** and underlined scores indicate the best and second-best performing methods for each model, respectively, with the exception of the full attention baseline.

Method	Llama-3.1-8B	Qwen2.5-7B-1M
Full	28.83	35.19
Minference	29.03	<u>34.19</u>
FlexPrefill	27.24	27.83
XAttention	<u>29.62</u>	<u>34.19</u>
MeanPooling	29.42	26.24
PBS-Attn	29.82	34.39

Efficiency Results To best evaluate the real-world practicality of the sparse attention methods, we measure the end-to-end time to first token (TTFT) on sequence lengths ranging from 8K to 512K. As shown in Figure 3, PBS-Attn achieves the highest speedup across all context lengths, whereas most competing methods only excel within a limited range. For instance, Minference does not show a speedup over FlashAttention until 128k, and the efficiency gains of XAttention stagnate after 128K. Although FlexPrefill matches the speedup of PBS-Attn in most cases, it suffers from a significant quality drop as shown in Table 1 and 2. In contrast, PBS-Attn consistently delivers the best performance, reaching a $2.75 \times$ end-to-end speedup at 256K, demonstrating its superior

practicality and robustness. To analyze the permutation overhead in PBS-Attn, we further conduct a detailed benchmarking study in Appendix C.

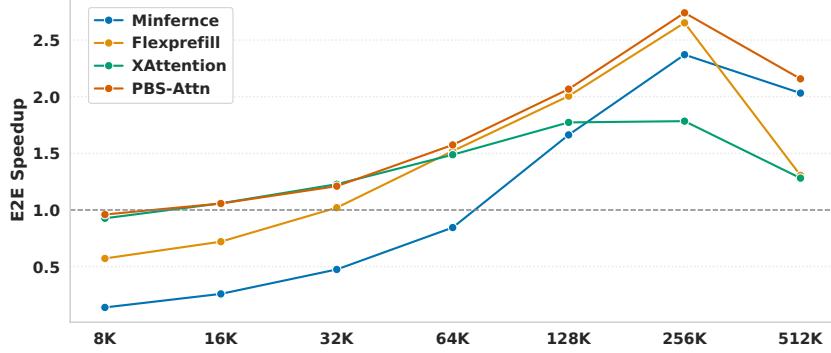


Figure 3: Speedup of various methods relative to FlashAttention, measured by time to first token (TTFT) on LongBenchv2 across various sequence lengths. To accommodate longer sequences under memory constraints, we employ tensor parallelism with `tp_size` of 2 and 8 for the 256K and 512K contexts, respectively.

4.3 ABLATION STUDIES AND ANALYSIS

Effect of Permutation As illustrated in Figure 4, query-aware key permutation consistently increases block-level sparsity by a noticeable margin. For instance, it achieves a 7% absolute sparsity improvement at a context length of 8K, and this gain continues to increase as the context length scales, highlighting the permutation’s effectiveness.

Permutation Target Analysis To analyze the effect of permutation on queries, we propose a key-aware query permutation approach. However, the attention distribution of queries over keys is often less structured than that of keys over queries. We therefore employ a straightforward strategy that clusters queries which attend to similar keys within a given segment. Specifically, we first compute a set of centroids by calculating block-averaged keys, denoted as \bar{K} . Each centroid is defined as $\bar{K}_i = \text{MeanPool}(\bar{K}_{[(i-1)B+1:iB]})$ for $i = 1, \dots, T_c$. We then determine cluster assignments by computing the cosine similarity between each query and these centroids. Within each segment, queries are assigned greedily based on their similarity to the centroids. We evaluate the effect of the permutation target and order in Figure 5a. The results indicate that permuting both queries and keys brings no noticeable improvements, regardless of the order. Permuting queries offers a marginal improvement over permuting keys in the performance-density trade-off, but it can be less efficient considering the overhead in models with Grouped-Query Attention (GQA) (Ainslie et al., 2023), which have multiple times more query heads than key heads. Accordingly, we exclusively adopt query-aware key permutation in our main method.

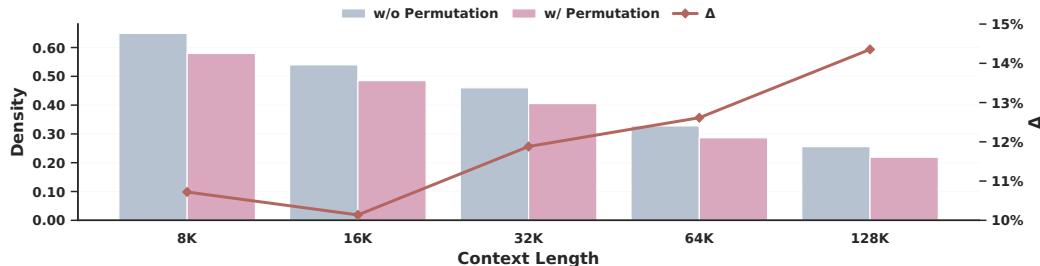


Figure 4: Block-level density on various context lengths with and without permutation. A relative sparsity improvement Δ is calculated.

Effect of Segment Size Segment size S plays a crucial role in segmented permutation, where tokens are permuted within the corresponding segments to maintain inter-segment causality. Intuitively, a larger segment size S takes into account more tokens during sorting, thereby enhancing block-level sparsity; however, it would also include more blocks in the on-diagonal segments, which can not be skipped during computation to avoid breaking causality. Figure 5b illustrates how the segment size, S , affects the performance-density trade-off. A larger S flattens the trade-off curve, indicating that segmented permutation effectively clusters key tokens, allowing the model to maintain high performance even at high levels of block-level sparsity. However, this benefit diminishes at lower sparsity levels, as the wide on-diagonal segments contain a large number of blocks that must be computed, limiting block-level sparsity.

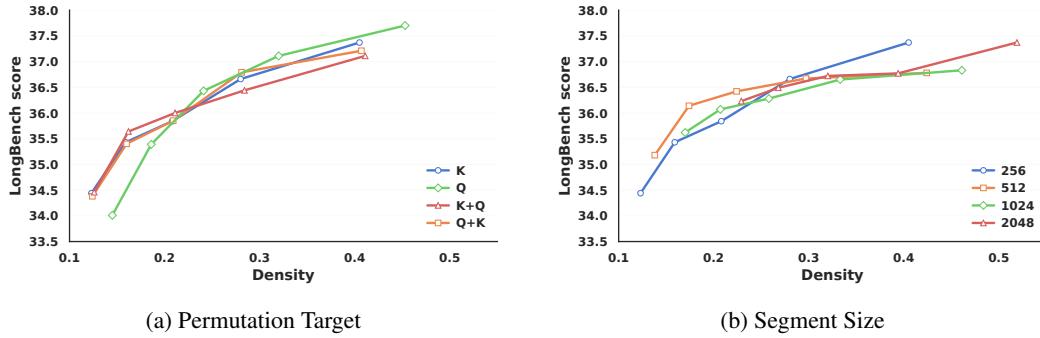


Figure 5: LongBench score vs. average block-level density at a context length of 32K.

5 RELATED WORK

Long-context LLMs As the capabilities of modern Large Language Models (LLMs) continue to advance, the expansion of their context length has become an inevitable and critical trend. A longer context window enables models to process a greater volume of information, which in turn facilitates more sophisticated context engineering (Mei et al., 2025) and agentic use (Wang et al., 2024). Extensive research has focused on extending the context length of LLMs through methods spanning data curation to post-training strategies (Liu et al., 2025). These advancements have culminated in models with context lengths up to millions of tokens (Yang et al., 2025a).

Sparse Attention The quadratic growth in memory and computational requirements of the attention mechanism has been a bottleneck for scaling LLM context lengths. Sparse attention has emerged as a promising solution, leveraging the inherent sparsity in attention patterns to drastically reduce this overhead. These methods can accelerate different stages of inference, such as prefilling, decoding, or both. StreamingLLM (Xiao et al., 2024) first identifies the attention sink phenomenon in LLMs, proposing to capture a majority of the attention mass with initial and recent tokens. NSA (Yuan et al., 2025) and MoBA (Lu et al., 2025) further incorporate sparse attention into the training stage, accelerating both prefilling and decoding. Methods like H2O (Zhang et al., 2023), can accelerate the decoding speed by exploiting the attention pattern after prefilling. Closely related to this work, various methods are proposed to accelerate the compute-bounded prefilling process. For example, Minference (Jiang et al., 2024) recognizes attention patterns in a pre-computed manner. More recent works tend to perform attention pattern recognition on-the-fly. For instance, FlexPrefill (Lai et al., 2025) utilizes divergence to classify the attention pattern, XAttention (Xu et al., 2025) adopts an antidiagonal scoring metric to weight each block, and SpargeAttention (Zhang et al., 2025) accounts the intra-block similarity into the selection criterion. However, these methods primarily focus on developing better block selection algorithms, while our work is orthogonal: we focus on rearranging the attention matrix to create a structure that inherently increases block-level sparsity.

Attention with Token Permutation Concurrent with our work, methods like SVG2 (Yang et al., 2025b) and PAROAttention (Zhao et al., 2025) show promise in accelerating visual generation models like Diffusion Transformers (Peebles & Xie, 2023), but their reliance on bidirectional attention

makes them incompatible with the causal constraints of auto-regressive LLMs. PBS-Attn accelerates this by introducing a segmented permutation strategy, explicitly preserving inter-segment causality.

6 CONCLUSION

In this work, we formalize the permutation properties of the attention mechanism and leverage them to improve block-level sparsity. We introduce Permuted Block-Sparse Attention (PBS-Attn), a plug-and-play method that employs a novel segmented permutation strategy to preserve inter-segment causality while reordering tokens within each segment. Our method achieves an end-to-end prefilling speedup of up to $2.75\times$ with minimal performance degradation, demonstrating a promising path toward more efficient long-context LLMs.

7 ETHICS STATEMENT

Our work focuses on improving the computational efficiency of large language models. We believe this research carries positive ethical implications. By reducing the computational resources required for processing long sequences, our method contributes to lowering the energy consumption and carbon footprint associated with training and deploying large-scale AI models. This can also enhance the accessibility of advanced AI technologies, enabling researchers and developers with limited resources to contribute to the field and innovate responsibly.

8 REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our work, we commit to making our research as transparent and accessible as possible.

Code All code used for our experiments, including the implementation of Permuted Block-Sparse Attention (PBS-Attn) and the custom permuted-FlashAttention kernel, will be made publicly available after the reviewing period. The repository will include scripts to run the evaluations and detailed instructions for setup.

Models The experiments were conducted using publicly available state-of-the-art large language models: Llama-3.1-8B (128K) and Qwen-2.5-7B-1M (1M), available for downloading from platforms like HuggingFace.

Datasets We used two publicly available and widely recognized benchmarks for long-context language understanding: LongBench and LongBenchv2, available for downloading from platforms like HuggingFace.

Experimental Setup All experiments were performed on NVIDIA H100 80GB GPUs. Key implementation details are stated in Section 4.1.

REFERENCES

- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghvi. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL <https://arxiv.org/abs/2305.13245>.
- Anthropic. System Card: Claude Opus 4 & Claude Sonnet 4, May 2025. URL <https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf>.
- Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding, 2024. URL <https://arxiv.org/abs/2308.14508>.
- Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2025. URL <https://arxiv.org/abs/2412.15204>.
- Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. URL <https://arxiv.org/abs/2405.21060>.
- Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.
- 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>.
- Gemini Team, Google. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities, 2025. URL https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenев, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind That-tai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jong-soo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasudevan Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yearly, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogochhev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan,

Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparth, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vitor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuwei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghatham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo

Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wencheng Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhao-duo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL <https://arxiv.org/abs/2312.00752>.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024. URL <https://arxiv.org/abs/2407.02490>.

Yibo Jin, Tao Wang, Huimin Lin, Mingyang Song, Peiyang Li, Yipeng Ma, Yicheng Shan, Zhengfan Yuan, Cailong Li, Yajing Sun, Tiandeng Wu, Xing Chu, Ruizhi Huan, Li Ma, Xiao You, Wenting Zhou, Yunpeng Ye, Wen Liu, Xiangkun Xu, Yongsheng Zhang, Tiantian Dong, Jiawei Zhu, Zhe Wang, Xijian Ju, Jianxun Song, Haoliang Cheng, Xiaojing Li, Jiandong Ding, Hefei Guo, and Zhengyong Zhang. P/d-serve: Serving disaggregated large language model at scale, 2024. URL <https://arxiv.org/abs/2408.08147>.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL <https://arxiv.org/abs/2006.16236>.

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>.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023. URL <https://arxiv.org/abs/2310.01889>.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, Yuanxing Zhang, Zhuo Chen, Hangyu Guo, Shilong Li, Ziqiang Liu, Yong Shan, Yifan Song, Jiayi Tian, Wenhao Wu, Zhejian Zhou, Ruijie Zhu, Junlan Feng, Yang Gao, Shizhu He, Zhoujun Li, Tianyu Liu, Fanyu Meng, Wenbo Su, Yingshui Tan, Zili Wang, Jian Yang, Wei Ye, Bo Zheng, Wangchunshu Zhou, Wenhao Huang, Sujian Li, and Zhaoxiang Zhang. A comprehensive survey on long context language modeling, 2025. URL <https://arxiv.org/abs/2503.17407>.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms, 2025. URL <https://arxiv.org/abs/2502.13189>.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A survey of context engineering for large language models, 2025. URL <https://arxiv.org/abs/2507.13334>.

OpenAI. GPT-5 System Card, August 2025. URL <https://cdn.openai.com/gpt-5-system-card.pdf>.

William Peebles and Saining Xie. Scalable diffusion models with transformers, 2023. URL <https://arxiv.org/abs/2212.09748>.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models, 2023. URL <https://arxiv.org/abs/2309.00071>.

- Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation, 2022. URL <https://arxiv.org/abs/2108.12409>.
- Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL <https://arxiv.org/abs/2104.09864>.
- Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019. URL <https://api.semanticscholar.org/CorpusID:184488182>.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL <https://arxiv.org/abs/1706.03762>.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jia-akai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), March 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL <http://dx.doi.org/10.1007/s11704-024-40231-1>.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL <https://arxiv.org/abs/2309.17453>.
- Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring, 2025. URL <https://arxiv.org/abs/2503.16428>.
- An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. Qwen2.5-1m technical report, 2025a. URL <https://arxiv.org/abs/2501.15383>.
- Shuo Yang, Haocheng Xi, Yilong Zhao, Muyang Li, Jintao Zhang, Han Cai, Yujun Lin, Xiuyu Li, Chenfeng Xu, Kelly Peng, Jianfei Chen, Song Han, Kurt Keutzer, and Ion Stoica. Sparse videogen2: Accelerate video generation with sparse attention via semantic-aware permutation, 2025b. URL <https://arxiv.org/abs/2505.18875>.
- Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025c. URL <https://arxiv.org/abs/2412.06464>.
- Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length, 2025d. URL <https://arxiv.org/abs/2406.06484>.
- Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL <https://arxiv.org/abs/2502.11089>.
- Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spageattention: Accurate and training-free sparse attention accelerating any model inference, 2025. URL <https://arxiv.org/abs/2502.18137>.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H₂O: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL <https://arxiv.org/abs/2306.14048>.
- Tianchen Zhao, Ke Hong, Xinhao Yang, Xuefeng Xiao, Huixia Li, Feng Ling, Ruiqi Xie, Siqi Chen, Hongyu Zhu, Yichong Zhang, and Yu Wang. Paroattention: Pattern-aware reordering for efficient sparse and quantized attention in visual generation models, 2025. URL <https://arxiv.org/abs/2506.16054>.

A PROOFS OF PERMUTATION PROPERTIES

A.1 PROOF OF LEMMA 3.1

Lemma A.1 (Key-Value Pair Permutation Invariance). *The attention mechanism is invariant to the order of the source sequence, provided that the key-value pairings are maintained.*

Formally, let $\mathbf{P}_\pi \in \{0, 1\}^{M \times M}$ be a permutation matrix that reorders the rows of a matrix according to a permutation π on the index set $\{1, \dots, M\}$. The following identity holds:

$$\text{Attention}(\mathbf{Q}, \mathbf{P}_\pi \mathbf{K}, \mathbf{P}_\pi \mathbf{V}) = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad (12)$$

Proof. Let $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ and $\mathbf{O}' = \text{Attention}(\mathbf{Q}, \mathbf{P}_\pi \mathbf{K}, \mathbf{P}_\pi \mathbf{V})$. Our goal is to show that $\mathbf{O} = \mathbf{O}'$. We will prove this by showing that their corresponding row vectors, \mathbf{o}_i and \mathbf{o}'_i , are equal for any arbitrary row index $i \in \{1, \dots, N\}$.

Let $\mathbf{A} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}$ and $\mathbf{W} = \text{softmax}(\mathbf{A})$ (we use \mathbf{W} instead of \mathbf{P} as in Eq.1 to avoid confusion). The i -th row of the original output is given by:

$$\mathbf{o}_i = \sum_{j=1}^M \mathbf{W}_{ij} \mathbf{v}_j$$

Now, let $\mathbf{K}' = \mathbf{P}_\pi \mathbf{K}$ and $\mathbf{V}' = \mathbf{P}_\pi \mathbf{V}$. The score matrix for \mathbf{O}' is $\mathbf{A}' = \frac{\mathbf{Q}(\mathbf{K}')^T}{\sqrt{d}} = \frac{\mathbf{Q}\mathbf{K}^T \mathbf{P}_\pi^T}{\sqrt{d}} = \mathbf{A} \mathbf{P}_\pi^T$. Let $\mathbf{W}' = \text{softmax}(\mathbf{A}')$.

The (i, j) -th element of \mathbf{A}' is $\mathbf{A}'_{ij} = \sum_{l=1}^M \mathbf{A}_{il} (\mathbf{P}_\pi^T)_{lj} = \mathbf{A}_{i, \pi^{-1}(j)}$. The denominator for the softmax computation on the i -th row of \mathbf{A}' is:

$$\sum_{l=1}^M \exp(\mathbf{A}'_{il}) = \sum_{l=1}^M \exp(\mathbf{A}_{i, \pi^{-1}(l)})$$

Since π^{-1} is a bijection on $\{1, \dots, M\}$, this summation is a reordering of the terms $\sum_{k=1}^M \exp(\mathbf{A}_{ik})$, which is the denominator for the i -th row of the original weights \mathbf{W} .

Thus, the (i, j) -th element of the new weight matrix \mathbf{W}' is:

$$\mathbf{W}'_{ij} = \frac{\exp(\mathbf{A}'_{ij})}{\sum_{l=1}^M \exp(\mathbf{A}'_{il})} = \frac{\exp(\mathbf{A}_{i, \pi^{-1}(j)})}{\sum_{k=1}^M \exp(\mathbf{A}_{ik})} = \mathbf{W}_{i, \pi^{-1}(j)}$$

The i -th row of the new output \mathbf{O}' is a weighted sum of the rows of $\mathbf{V}' = \mathbf{P}_\pi \mathbf{V}$. The j -th row of \mathbf{V}' is $\mathbf{v}'_j = \mathbf{v}_{\pi^{-1}(j)}$. Therefore:

$$\mathbf{o}'_i = \sum_{j=1}^M \mathbf{W}'_{ij} \mathbf{v}'_j = \sum_{j=1}^M \mathbf{W}_{i, \pi^{-1}(j)} \mathbf{v}_{\pi^{-1}(j)}$$

Let $k = \pi^{-1}(j)$. Since π^{-1} is a bijection, summing over all $j \in \{1, \dots, M\}$ is equivalent to summing over all $k \in \{1, \dots, M\}$. By this change of variables, we have:

$$\mathbf{o}'_i = \sum_{k=1}^M \mathbf{W}_{ik} \mathbf{v}_k = \mathbf{o}_i$$

Since $\mathbf{o}'_i = \mathbf{o}_i$ for an arbitrary i , the matrices \mathbf{O}' and \mathbf{O} are identical. \square

A.2 PROOF OF LEMMA 3.2

Lemma A.2 (Query Permutation Equivariance). *The attention mechanism is equivariant with respect to permutations of the query sequence.*

Formally, let $\mathbf{P}_\sigma \in \{0, 1\}^{N \times N}$ be a permutation matrix that reorders the rows of a matrix according to a permutation σ on the index set $\{1, \dots, N\}$. The following relationship holds:

$$\text{Attention}(\mathbf{P}_\sigma \mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{P}_\sigma \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad (13)$$

Proof. Let $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ and $\mathbf{O}' = \text{Attention}(\mathbf{P}_\sigma \mathbf{Q}, \mathbf{K}, \mathbf{V})$. We want to show that $\mathbf{O}' = \mathbf{P}_\sigma \mathbf{O}$.

Let $\mathbf{A} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}$ and $\mathbf{W} = \text{softmax}(\mathbf{A})$, such that $\mathbf{O} = \mathbf{W}\mathbf{V}$. The score matrix for \mathbf{O}' is $\mathbf{A}' = \frac{(\mathbf{P}_\sigma \mathbf{Q})\mathbf{K}^T}{\sqrt{d}} = \mathbf{P}_\sigma \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) = \mathbf{P}_\sigma \mathbf{A}$. Let $\mathbf{W}' = \text{softmax}(\mathbf{A}')$.

The softmax function operates independently on each row. Let $(\mathbf{X})_i$ denote the i -th row of a matrix \mathbf{X} . Left-multiplication by \mathbf{P}_σ permutes the rows of \mathbf{A} , such that the i -th row of \mathbf{A}' is the $\sigma^{-1}(i)$ -th row of \mathbf{A} : $(\mathbf{A}')_i = (\mathbf{A})_{\sigma^{-1}(i)}$. Applying the softmax function, the i -th row of \mathbf{W}' is:

$$(\mathbf{W}')_i = \text{softmax}((\mathbf{A}')_i) = \text{softmax}((\mathbf{A})_{\sigma^{-1}(i)})$$

This resulting vector is identical to the $\sigma^{-1}(i)$ -th row of the original weight matrix \mathbf{W} . Thus, $(\mathbf{W}')_i = (\mathbf{W})_{\sigma^{-1}(i)}$. This equality for all rows i implies that the entire matrix \mathbf{W}' is a row-permuted version of \mathbf{W} , i.e., $\mathbf{W}' = \mathbf{P}_\sigma \mathbf{W}$.

Now we can write the output \mathbf{O}' as:

$$\mathbf{O}' = \mathbf{W}'\mathbf{V} = (\mathbf{P}_\sigma \mathbf{W})\mathbf{V}$$

By the associativity of matrix multiplication, we have:

$$\mathbf{O}' = \mathbf{P}_\sigma(\mathbf{W}\mathbf{V}) = \mathbf{P}_\sigma \mathbf{O}$$

This completes the proof. \square

A.3 PROOF OF THEOREM 3.3

Theorem A.3 (Attention Permutation Invariance under Inverse Transformation). *If the queries are permuted by \mathbf{P}_σ and the key-value pairs are permuted by \mathbf{P}_π , the resulting output is a permuted version of the original output. Applying the inverse of the query permutation recovers the original, unpermuted output. Formally:*

$$\mathbf{P}_\sigma^T \text{Attention}(\mathbf{P}_\sigma \mathbf{Q}, \mathbf{P}_\pi \mathbf{K}, \mathbf{P}_\pi \mathbf{V}) = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \quad (14)$$

Proof. We prove the theorem by showing that the left-hand side (LHS) of the equation simplifies to the right-hand side (RHS) through sequential application of the preceding lemmas.

$$\begin{aligned} \text{LHS} &= \mathbf{P}_\sigma^T \text{Attention}(\mathbf{P}_\sigma \mathbf{Q}, \mathbf{P}_\pi \mathbf{K}, \mathbf{P}_\pi \mathbf{V}) \\ &= \mathbf{P}_\sigma^T \text{Attention}(\mathbf{P}_\sigma \mathbf{Q}, \mathbf{K}, \mathbf{V}) && \text{by Lemma 3.1} \\ &= \mathbf{P}_\sigma^T (\mathbf{P}_\sigma \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})) && \text{by Lemma 3.2} \\ &= (\mathbf{P}_\sigma^T \mathbf{P}_\sigma) \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) && \text{by associativity} \\ &= I \cdot \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) && \text{since } \mathbf{P}_\sigma \text{ is orthogonal} \\ &= \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ &= \text{RHS} \end{aligned}$$

The final expression is identical to the right-hand side, which concludes the proof. \square

B BLOCK SELECTION

B.1 BLOCK SELECTION IN PBS-ATTN

We use a mean pooling strategy and block-wise attention to estimate the importance of each key block. This method is also used for unpermuted sequences, serving as a strong baseline denoted as MeanPooling in the main paper. Here we detail the implementation of MeanPooling selection in Algorithm 2. Note that for the baseline MeanPooling, \mathbf{Q}' and \mathbf{K}' remain unpermuted as $\mathbf{Q}' = \mathbf{Q}$ and $\mathbf{K}' = \mathbf{K}$. The causal mask \mathbf{C} is a upper triangular matrix with entries set to $-\infty$. If segmented permutation is applied, this mask also includes the on-diagonal segments (as in Figure 1), to ensure valid intra-segment attention post-permutation.

Algorithm 2 MeanPooling Block Selection

Require: Query matrix $\mathbf{Q}' \in \mathbb{R}^{N \times d}$, Key matrix $\mathbf{K}' \in \mathbb{R}^{N \times d}$, block size B , attention score threshold τ , causal mask $\mathbf{C} \in \{0, -\infty\}^{\lceil N/B \rceil \times \lceil N/B \rceil}$.

Ensure: Block selection mask $\mathbf{M} \in \{0, 1\}^{\lceil N/B \rceil \times \lceil N/B \rceil}$.

- 1: Divide \mathbf{Q}' , \mathbf{K}' into blocks of size B : $\{\mathbf{Q}'_i\}_{i=1}^{T_r}$, $\{\mathbf{K}'_j\}_{j=1}^{T_c}$, where $T_r = T_c = \lceil N/B \rceil$.
- 2: Compute pooled queries: $\bar{\mathbf{Q}}_i = \text{MeanPool}(\mathbf{Q}'_i)$ for $i = 1, \dots, T_r$.
- 3: Compute pooled keys: $\bar{\mathbf{K}}_j = \text{MeanPool}(\mathbf{K}'_j)$ for $j = 1, \dots, T_c$.
- 4: Form pooled matrices $\bar{\mathbf{Q}} \in \mathbb{R}^{T_r \times d}$ and $\bar{\mathbf{K}} \in \mathbb{R}^{T_c \times d}$.
- 5: Compute block scores: $\mathbf{S}_{\text{block}} = \text{softmax}(\bar{\mathbf{Q}} \bar{\mathbf{K}}^T / \sqrt{d} + \mathbf{C})$.
- 6: Initialize $\mathbf{M} = \mathbf{0}$.
- 7: **for** $i = 1$ to T_r **do**
- 8: Get scores for query block i : $\mathbf{a}_i = \mathbf{S}_{\text{block}}[i, 1 : i]$.
- 9: Sort scores and get original indices: $\mathbf{o}_i = \text{argsort}(-\mathbf{a}_i)$.
- 10: Compute cumulative sum on sorted scores: $\mathbf{c}_i = \text{cumsum}(\mathbf{a}_i[\mathbf{o}_i])$.
- 11: Find number of blocks to select: $k = \min(\{j \mid \mathbf{c}_i[j] \geq \tau\} \cup \{i\})$.
- 12: Get indices of blocks to select: $\mathcal{J} = \mathbf{o}_i[1 : k]$.
- 13: Set $\mathbf{M}[i, j] = 1$ for all $j \in \mathcal{J}$.
- 14: **end for**
- 15: **return** \mathbf{M} .

B.2 PBS-ATTN WITH EXISTING BLOCK SELECTION ALGORITHMS

In the main paper, we use a simple mean pooling strategy for block selection in block-sparse attention, as detailed in Section B.1, and show that permutation can increase block-level sparsity under this naive mean pooling strategy (Section 4.3). In this section, we further demonstrate that advanced block selection algorithms (e.g. XAttention) can also benefit from permutation.

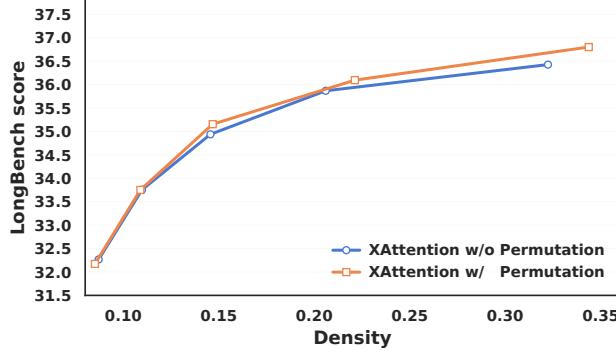


Figure 6: Longbench score vs. average block-level density at a context length of 32k of XAttention selection with and without permutation.

As shown in Figure 6, XAttention selection can also benefit from the sparsity improvements of permutation, achieving a better trade-off between performance and sparsity.

C ANALYSIS ON THE PERMUTATION OVERHEAD

As shown in Figures 7a and 7b, the permutation overhead in PBS-Attn is negligible compared to the main attention computation time, especially at longer context lengths. For instance, at a context length of 128K, permutation introduces an overhead of only 4% relative to the block attention computation time and just 1.3% compared to FlashAttention. While permuting queries introduces a slightly higher overhead than permuting keys, this difference diminishes as the context length increases. However, query permutation can also result in lower block-level sparsity than key permutation under the same settings, leading to higher attention computation time.

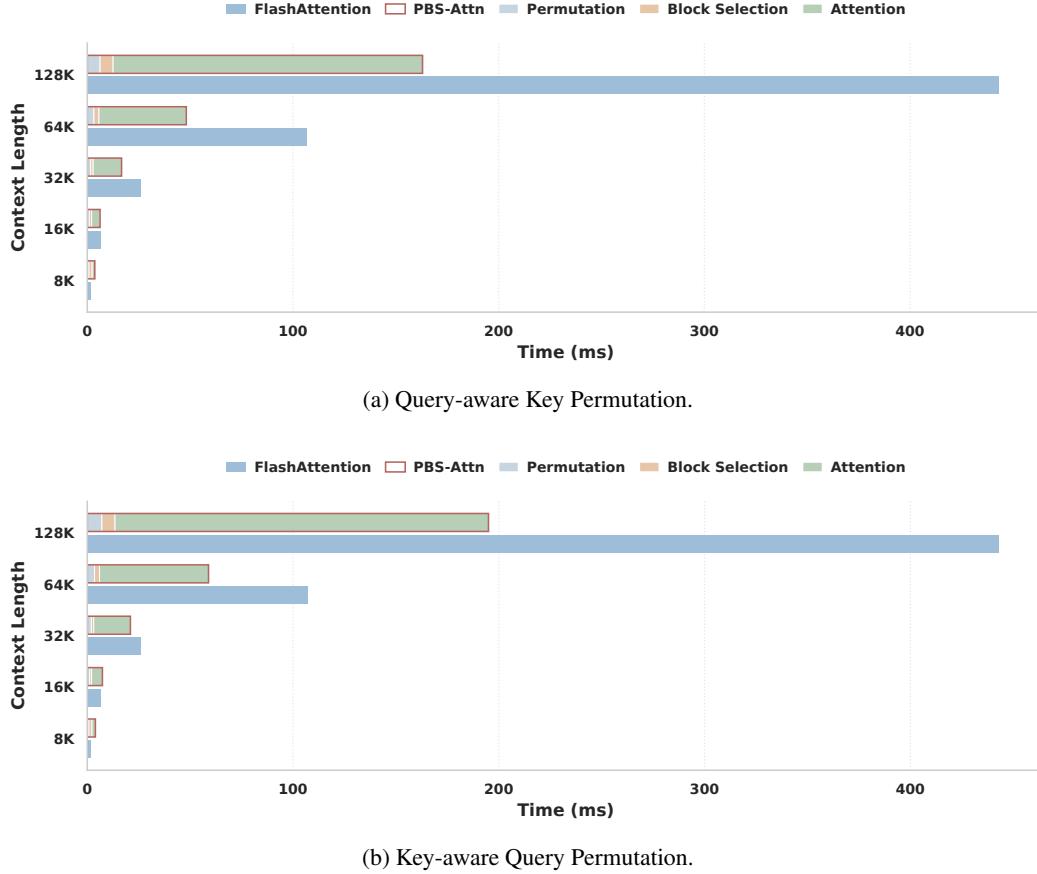


Figure 7: Detailed benchmarking results of PBS-Attn vs. FlashAttention.

D VISUALIZATION OF PERMUTATION

In this section, we provide more visualizations of the permutation effect on both Llama-3.1-8B (Figure 8) and Qwen-2.5-7B-1M (Figure 9).

E USE OF LARGE LANGUAGE MODELS

During the preparation of this work, we utilized large language models (LLMs) to assist with code development and manuscript writing. Specifically, their applications included improving the grammar and clarity of the text, as well as assisting in code completion.

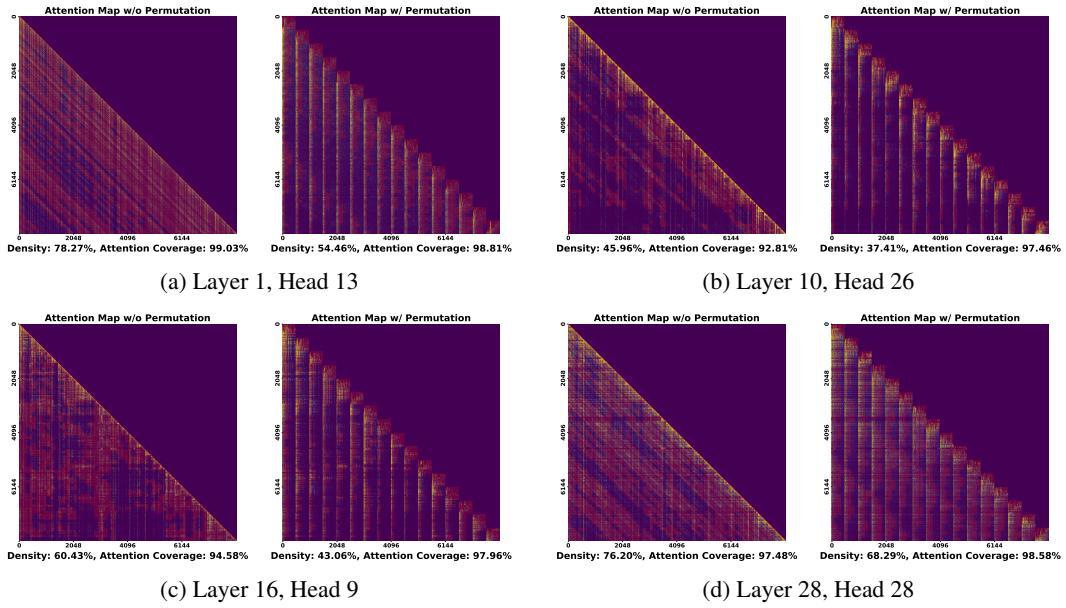


Figure 8: Permutation visualizations of Llama-3.1-8B.

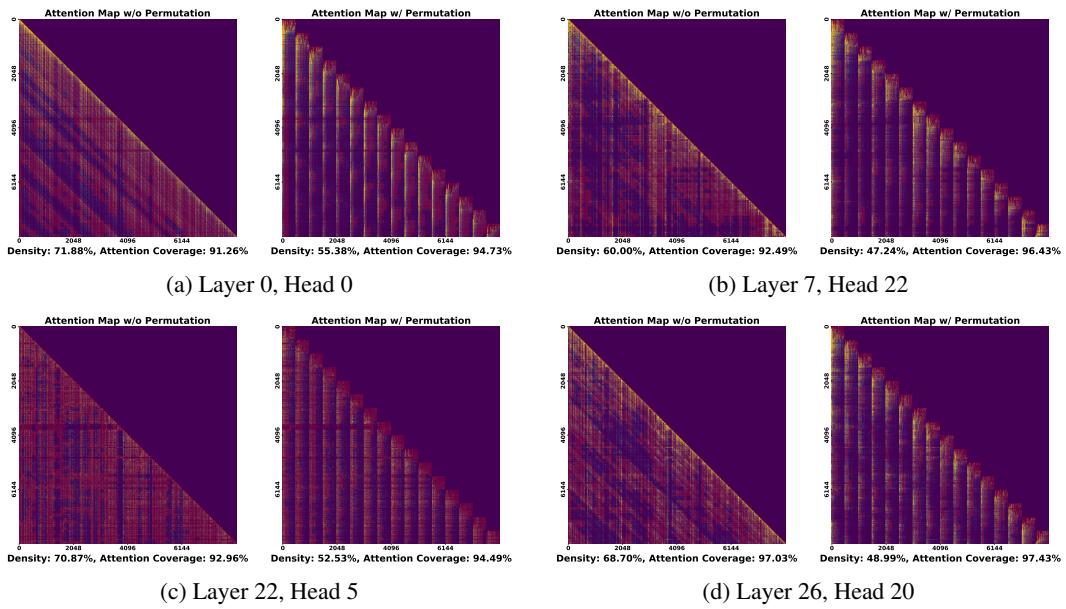


Figure 9: Permutation visualizations of Qwen-2.5-7B-1M.