RoBSA: RoPE-based Blockwise Sparse Multi-head Latent Attention

Xinyu Shi[1] **Kairong Luo**[1] **Zhen Zheng**[3] **Wenguang Chen**[1,2†]
[1]Department of Computer Science & Technology, Tsinghua University
[2]Tsinghua Shenzhen International Graduate School, Tsinghua University
[3]Microsoft

Abstract

Large Language Models (LLMs) have rapidly advanced in recent years, scaling up in both parameter count and context length. However, as context windows extend from thousands to hundreds of thousands of tokens, attention computation becomes the dominant source of memory usage and runtime in decoding stages, severely limiting the efficiency and scalability of long-context LLMs. Sparse attention has emerged as a promising solution, reducing complexity by computing attention over only a subset of context tokens. However, the sparse attention for Multi-head Latent Attention (MLA) which is a variant of standard MHA is rarely studied. In this paper, we introduce RoPE-based Blockwise Sparse Attention (RoBSA), a method designed specifically for MLA during the decoding stage of model inference. RoBSA leverages the decoupled nature of RoPE within MLA to implement token selection in a blockwise manner. RoBSA is a lightweight, training-free, and layer-aware algorithm that can be integrated in a plug-and-play fashion. Our method significantly reduces end-to-end inference latency in the decoding stage by up to $2.55\times$ with minimal accuracy loss compared to full attention in long-context scenarios for very large models.

1 Introduction

In recent years, Large Language Models (LLMs) have advanced significantly, primarily by increasing their parameter counts and extending their context windows (OpenAI et al., 2024; Yang et al., 2025a). State-of-the-art models now feature trillions of parameters and can process millions of tokens, allowing them to analyze entire books or extensive codebases in a single pass (Team et al., 2025; Comanici et al., 2025). This expanded capacity is crucial for complex reasoning, coherent multi-turn dialogue, and fine-tuning with methods like reinforcement learning, which often depend on understanding long-range dependencies within the context.

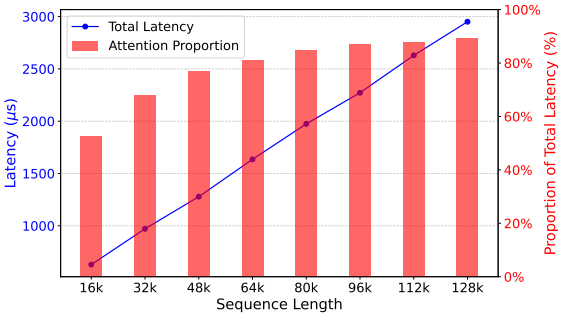


Figure 1: The attention kernel's computation time as a proportion of the total decoding time for DeepSeek-R1. With a 128k token context, attention accounts for over 90% of the latency, highlighting it as the primary bottleneck.

However, processing such long contexts presents a major computational challenge. The bottleneck is the self-attention mechanism, a core component of the Transformer architecture (Zhou et al., 2024). Its computational cost scales linearly with the sequence length during decoding stage, making inference prohibitively slow and memory-intensive for long sequences. As shown in Figure 1, when the context length extends to 128k tokens, the attention computation alone can consume over 90% of the total inference time. This inefficiency severely limits the practical deployment of long-context LLMs in real-world applications.

To mitigate this bottleneck, researchers have developed various efficiency-focused techniques (Zhou et al., 2024). These include architectural modifications like Multi-Query Attention (MQA) (Shazeer, 2019), Grouped-Query Attention (GQA) (Ainslie et al., 2023), and Multi-head Latent Attention (MLA) (DeepSeek-AI et al., 2024), which reduce the size of the Key-Value cache. Some attempts to use linear

†Corresponding author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1028–1044

attention (Yang et al., 2025b), to reduce attention compute complexity. Other approaches focus on approximating the attention matrix through methods like sparse attention (Child et al., 2019), or on model compression techniques such as quantization (Frantar et al., 2023) and weight pruning (Frantar and Alistarh, 2023).

Despite these advances, a critical gap remains in applying these solutions to the latest generation of large-scale models. Many leading models, such as DeepSeek-V3 (DeepSeek-AI et al., 2025b), DeepSeek-R1 (DeepSeek-AI et al., 2025a), and Kimi-K2 (Team et al., 2025), utilize Multi-head Latent Attention (MLA), an architecture with unique features like a compressed latent space and partial application of Rotary Position Embeddings (RoPE) (Su et al., 2023). These design choices make it challenging to directly apply conventional sparse attention methods, which are often designed for standard attention mechanisms and may not be compatible with MLA's structure without significant modifications or retraining.

This paper introduces a key observation specific to the MLA architecture: *the attention scores computed using only the small portion of hidden dimensions to which RoPE is applied are highly similar to the scores computed using the full dimensions.* This finding suggests that these RoPE-applied dimensions can serve as an efficient and reliable proxy to identify the most important tokens in the context. Based on this insight, we propose a blockwise selection strategy. Instead of computing the full, costly attention matrix, we use the lightweight proxy scores to select a small subset of token blocks from the KV cache and perform attention only on them, drastically reducing computation.

Our method, named RoPE-based Blockwise Sparse Attention (RoBSA), delivers significant performance improvements. In extensive experiments on models with hundreds of billions of parameters, RoBSA achieves up to a 2.55x end-to-end speedup in long-context inference compared to the standard MLA implementation. This acceleration is achieved with minimal degradation in model accuracy, demonstrating the effectiveness and efficiency of our approach.

Our main contributions are threefold:

- We are the first to identify and analyze the strong similarity between attention scores derived from RoPE-applied dimensions and full dimensions within the MLA framework.
- We design RoBSA, a novel, training-free, and plug-and-play sparse attention algorithm that leverages this property for efficient blockwise attention computation.
- We conduct comprehensive experiments on massive-scale language models, demonstrating that RoBSA provides substantial inference speedups on various long-context benchmarks with negligible impact on accuracy.

2 Methodology

2.1 Preliminaries: Multi-head Latent Attention

Our work builds upon Multi-head Latent Attention (MLA), an efficient attention mechanism first introduced in DeepSeek-V2 (DeepSeek-AI et al., 2024). MLA is designed to reduce the size of the Key-Value (KV) cache during inference, which is a major bottleneck in long-context scenarios. It achieves this by compressing the keys and values into a shared low-rank latent space.

Let $\mathbf{h}_t \in \mathbb{R}^d$ be the input hidden state for the t-th token. MLA first projects this input into separate compressed latent vectors for the query ($\mathbf{c}_t^Q$) and the key-value pair ($\mathbf{c}_t^{KV}$). These latent vectors are then used to generate the final queries, keys, and values. The complete formulation is as follows:

First, the key $\mathbf{k}_t$ and value $\mathbf{v}_t$ are computed. The key is formed by concatenating a compressed component $\mathbf{k}_t^C$ and a RoPE component $\mathbf{k}_t^R$:

$$\mathbf{c}_t^{KV} = W^{DKV}\mathbf{h}_t \tag{1}$$
$$\mathbf{k}_t^C = W^{UK}\mathbf{c}_t^{KV} \tag{2}$$
$$\mathbf{k}_t^R = \mathrm{RoPE}(W^{KR}\mathbf{h}_t) \tag{3}$$
$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R] \tag{4}$$
$$\mathbf{v}_t^C = W^{UV}\mathbf{c}_t^{KV} \tag{5}$$

Here, $\mathbf{c}_t^{KV} \in \mathbb{R}^{d_c}$ is the compressed latent vector, with its dimension d_c being much smaller than the full hidden dimension. During inference, only the vectors $\mathbf{c}_t^{KV}$ and $\mathbf{k}_t^R$ need to be cached for each token, leading to significant memory savings. Similarly, the query $\mathbf{q}_t$ is computed:

$$\mathbf{c}_t^Q = W^{DQ}\mathbf{h}_t \tag{6}$$
$$\mathbf{q}_t^C = W^{UQ}\mathbf{c}_t^Q \tag{7}$$
$$\mathbf{q}_t^R = \mathrm{RoPE}(W^{QR}\mathbf{c}_t^Q) \tag{8}$$
$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R] \tag{9}$$

Finally, the attention output $\mathbf{o}_{t,i}$ for each head is calculated using the constructed queries, keys, and values:

$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \text{Softmax}_j \left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h^C + d_h^R}} \right) \mathbf{v}_{j,i}^C \quad (10)$$

where d_h^C and d_h^R are the dimensions per head for the compressed and RoPE parts, respectively, and $[\cdot;\cdot]$ denotes concatenation.

2.2 Key Observations and Hypothesis

The efficiency of Multi-head Latent Attention (MLA) can be significantly improved by exploiting the inherent sparsity in its attention mechanism. In this section, we first demonstrate that MLA's attention patterns are sparse. We then introduce our core insight: the small subset of dimensions that carry positional information via RoPE are disproportionately important for determining token relevance. This leads to our central hypothesis that attention scores computed using only these dimensions can serve as an effective, low-cost proxy for the full attention scores, which we then verify experimentally.

2.2.1 Sparsity in Multi-head Latent Attention

It is well-established that standard attention mechanisms are sparse, meaning each token attends strongly to only a small fraction of the preceding context. We observe that this property holds true for MLA as well. As shown in Figure 2, attention score heatmaps from a large model like DeepSeek-V3 reveal distinct sparse patterns that vary across different layers and heads. For instance, some heads exhibit a phenomenon known as the **token sink** (Xiao et al., 2024c), where attention is concentrated on the initial few tokens and the immediate local context. Other heads display **vertical patterns** (Jiang et al., 2024), where a few specific tokens scattered throughout the context receive high attention from all subsequent tokens. The diversity of these patterns highlights that static sparsity methods, such as a fixed-size local window, are inadequate for complex models. Therefore, a dynamic and fine-grained approach is required to effectively leverage this sparsity and accelerate inference.

To further quantify the sparsity of MLA, we calculate the cumulative distribution of attention scores across all layers of DeepSeek-V3, specifically focusing on the top $k\%$ of tokens with the highest attention scores. The results are illustrated in Figure 3. As shown in the figure, the attention scores in MLA exhibit a high degree of concentration: across nearly all layers, the top 5% tokens account for over 0.9 of the total attention score, while the top 10% tokens contribute more than 0.92. This quantitative evidence strongly demonstrates the inherent sparsity of MLA, implying that the vast majority of historical tokens have a negligible impact on the current decoding step, with only a few "key tokens" playing a decisive role in the attention mechanism.

2.2.2 The Critical Role of Position-Aware Dimensions

In Transformer architectures, positional embeddings are essential for providing information about token order. Without them, the self-attention mechanism becomes permutation-invariant; that is, shuffling the order of key and value tokens in the context would simply result in a correspondingly shuffled output, as shown in Equation 11, where $\mathbf{P}$ is a permutation matrix.

$$\text{Softmax}\left(\frac{\mathbf{q}\mathbf{k}^T}{d}\right)\mathbf{v} = \text{Softmax}\left(\frac{\mathbf{q}(\mathbf{P}\mathbf{k})^T}{d}\right)(\mathbf{P}\mathbf{v}) \quad (11)$$

MLA's decoupled design separates the position-agnostic components $(\mathbf{q}^C, \mathbf{k}^C)$ from the position-aware RoPE components $(\mathbf{q}^R, \mathbf{k}^R)$. The permutation invariance property applies only to the position-agnostic part. This inspires our key insight: since the RoPE-applied dimensions are the sole carriers of positional information, they must play a crucial role in identifying which tokens are most relevant based on their location in the sequence.

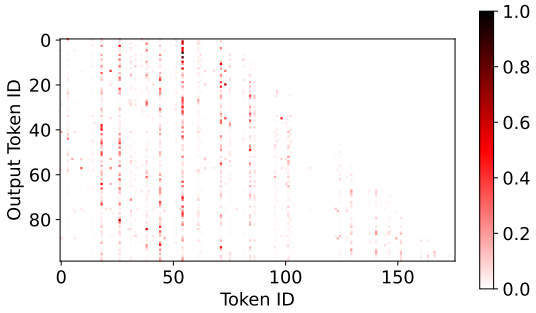
2.2.3 Hypothesis: RoPE Score as a Proxy for Full Attention Score

Based on this insight, we hypothesize that the attention score distribution computed using only the lightweight RoPE components strongly resembles the distribution of the full attention score. The full attention score, $\mathbf{S}$, is computed as:

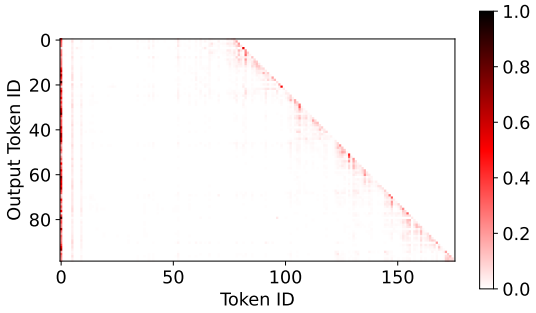
$$\mathbf{S} = \text{Softmax}\left(\frac{\mathbf{q}^C(\mathbf{k}^C)^T + \mathbf{q}^R(\mathbf{k}^R)^T}{\sqrt{d^C + d^R}}\right) \quad (12)$$

We define the *RoPE Attention Score*, $\mathbf{S}^R$, which uses only the position-aware components:

$$\mathbf{S}^R = \text{Softmax}\left(\frac{\mathbf{q}^R(\mathbf{k}^R)^T}{\sqrt{d^C + d^R}}\right) \quad (13)$$



(a) "Vertical" pattern (Layer 28, Head 54)



(b) "Token sink" pattern (Layer 1, Head 1)

Figure 2: Attention score heatmaps for different heads and layers in DeepSeek-V3, illustrating the diverse nature of sparsity across the model.

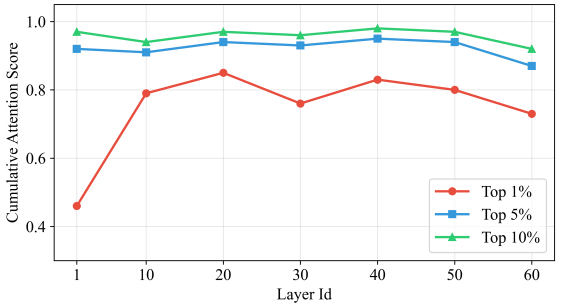


Figure 3: Cumulative sum of attention score of top k% tokens in each layer of DeepSeek-V3

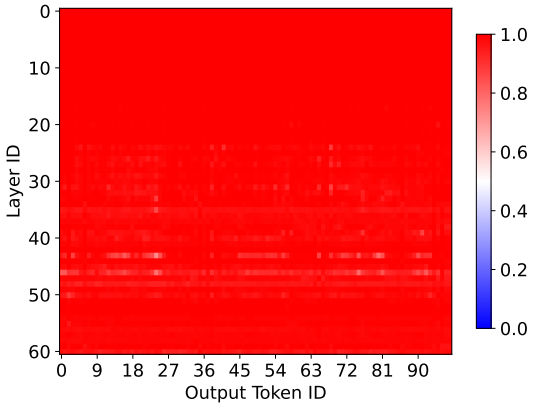


Figure 4: Cosine similarity between the full attention score ($\mathbf{S}$) and the RoPE-only attention score ($\mathbf{S}^R$) across all layers of DeepSeek-V3. The consistently high similarity (close to 1.0) validates our hypothesis.

If our hypothesis is correct, the computationally inexpensive $\mathbf{S}^R$ can be used to predict the "hotspots" in the full attention map, enabling a highly efficient method for selecting which tokens to attend to. To verify this, we calculated the cosine similarity between the full score vector $\mathbf{S}$ and the RoPE score vector $\mathbf{S}^R$ for every head and layer. As shown in Figure 4, the similarity is remarkably high (very close to 1.0) across nearly all layers, confirming our hypothesis. This strong correlation provides the foundation for our RoBSA algorithm, which uses these RoPE scores to guide a blockwise sparse attention mechanism.

2.3 RoBSA Algorithm Design

Based on our observations, we propose the **Ro**PE-**B**ased Blockwise **S**parse **A**ttention (RoBSA) algorithm. The design of RoBSA is guided by two primary strategies that directly address the findings from our analysis: (1) a layer-wise adaptive application that determines whether to use our RoPE-based approximation, and (2) an efficient blockwise selection mechanism that identifies the most salient tokens for the final attention computation. The complete inference pipeline is detailed in Algorithm 1.

2.3.1 Layer-wise Adaptive Application

Our analysis revealed that while the similarity between RoPE scores ($\mathbf{S}^R$) and full attention scores ($\mathbf{S}$) is high overall, it varies from layer to layer (as shown in Figure 5). Applying the RoPE-score approximation naively across all layers can lead to performance degradation. To address this, we introduce a layer-wise adaptive strategy. We first perform a one-time calibration process using a diverse dataset[1] to calculate the average similarity score for each layer.

Based on these scores, we partition the model's layers using a threshold θ_2:

- **RoPE-Sensitive Layers:** Layers where the average similarity is greater than θ_2. In these layers, the model's output is "sensitive" to the approximation, meaning the lightweight $\mathbf{S}^R$

[1]The dataset comprises 100 prompts from smoltalk (Allal et al., 2025), mmlu (Hendrycks et al., 2021), humaneval (Chen et al., 2021), gsm8k (Cobbe et al., 2021), and RULER (Hsieh et al., 2024).

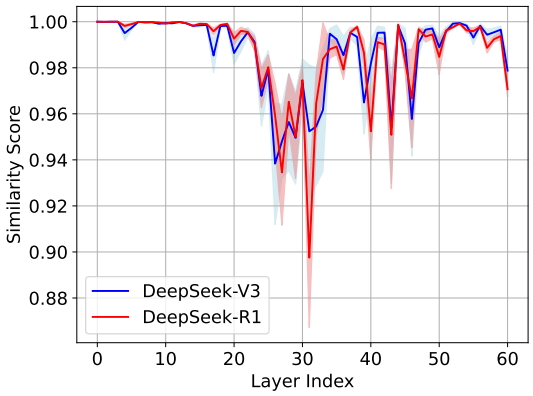


Figure 5: The average cosine similarity between the full attention score and the RoPE attention score for each layer, calculated over a calibration dataset. The variability across layers, especially in the middle, motivates our layer-wise partitioning strategy. The shaded areas represent the standard deviation across different samples.

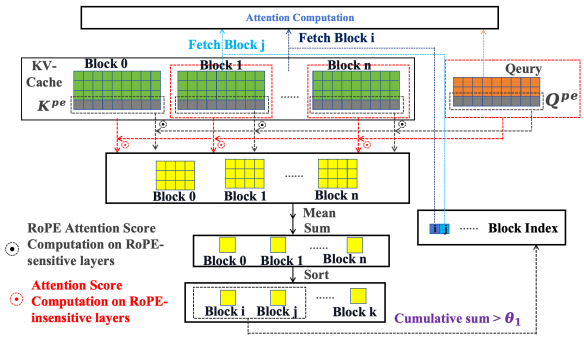


Figure 6: Workflow of RoBSA

is a reliable proxy for the full attention score. Our efficient selection method is applied here.

- **RoPE-Insensitive Layers:** Layers where the similarity is less than θ_2. Here, the model is "insensitive" to the approximation, and using $\mathbf{S}^R$ could be inaccurate. For these layers, we fall back to a more computationally intensive but accurate score calculation for block selection.

This static, one-time partitioning allows RoBSA to balance speed and accuracy by applying the approximation only where it is most effective.

2.3.2 Block-wise Token Selection

To make sparsity practical and avoid the latency of random memory access, RoBSA operates on contiguous blocks of the KV-Cache rather than individual tokens. This is a common technique in efficient attention design (Xiao et al., 2024a; Tang et al., 2024). The selection process at each decoding step is a multi-stage procedure, as formalized in Algorithm 1.

First, we perform an initial score calculation to estimate token importance. Based on our precomputed layer partition, we select the appropriate scoring function. For *RoPE-sensitive* layers, we compute the highly efficient proxy score $\mathbf{S}^R$ (Line 5). For the few *RoPE-insensitive* layers, we fall back to computing the full attention score $\mathbf{S}$ to ensure accuracy (Line 7).

Next, we aggregate these scores to derive a single importance value for each block. The initial score matrix has dimensions across heads and sequence length. We first reduce this matrix by averaging the scores across all attention heads, resulting in a single score for each token (Line 9). Then, we partition the KV cache along the sequence length dimension into fixed-size blocks and sum the scores of all tokens within each block (Line 10). This yields a short vector, $\mathbf{S}_b$, where each element represents the aggregated importance of its corresponding block.

Finally, we perform a dynamic block selection. The blocks are sorted in descending order based on their aggregated scores (Line 12). We then select the top-ranked blocks whose cumulative score sum exceeds a predefined threshold θ_1 (Line 13). This dynamic top-k method ensures that we capture a sufficient portion of the total attention mass, regardless of how concentrated or distributed it is. The full, exact attention computation is then performed only on the tokens within these selected blocks (Line 16), skipping the computation for the majority of the context and thus achieving significant speedup. The complete workflow is illustrated in Figure 6.

3 Experiments

3.1 Experiment Settings

Models and Evaluation Benchmarks We evaluate RoBSA on several long-context benchmarks to demonstrate its performance and feasibility. Our evaluation suite includes Needle-in-a-Haystack (NIAH) (Kamradt, 2024), LongBench v2 (Bai et al., 2025), InfiniteBench (Zhang et al., 2024), and LV-Eval (Yuan et al., 2024), where most prompts exceed 10,000 tokens. We employ two state-of-the-art, open-source models implemented with MLA: DeepSeek-V3 (DeepSeek-AI et al., 2025b) and DeepSeek-R1 (DeepSeek-AI et al., 2025a). DeepSeek-V3 is a Mixture-of-Experts

Algorithm 1 RoBSA for MLA Decoding

Input: block size B_h, $\mathbf{q}^N \in \mathbb{R}^{1\times n_h\times d^N}$, $\mathbf{q}^R \in \mathbb{R}^{1\times n_h\times d^R}$, blocked kv-cache $\mathbf{v}_t^C \in \mathbb{R}^{\lceil\frac{l}{B_h}\rceil\times B_h\times d^N}$, $\mathbf{k}^R \in \mathbb{R}^{\lceil\frac{l}{B_h}\rceil\times B_h\times d^R}$, $\mathbf{k} = [\mathbf{v}_t^C; \mathbf{k}^R] \in \mathbb{R}^{\lceil\frac{l}{B_h}\rceil\times B_h\times d}$, threshold θ_1, list of RoPE-insensitive layers $\mathbf{L}_s$, layer id l_i
Output: attention output $\mathbf{O}$
Stage 1: Compute blockwise attention score
if $l_i \notin \mathbf{L}_s$ **then** ▷ Layer is RoPE-Sensitive
  $\mathbf{S} \leftarrow \text{Softmax}[\frac{\mathbf{q}^R(\mathbf{k}^R)^T}{\sqrt{d^N+d^R}}]$, $\mathbf{S} \in \mathbb{R}^{1\times n_h\times l}$
else ▷ Layer is RoPE-Insensitive, fallback
  $\mathbf{S} \leftarrow \text{Softmax}[\frac{\mathbf{q}\mathbf{k}^T}{\sqrt{d^N+d^R}}]$, $\mathbf{S} \in \mathbb{R}^{1\times n_h\times l}$
end if
$\bar{\mathbf{S}} \leftarrow \frac{1}{n_h}\sum_{k=1}^{n_h}\mathbf{S}[:,k,:]$, $\bar{\mathbf{S}} \in \mathbb{R}^{1\times l}$ ▷ Mean over heads
$\mathbf{S}_b \leftarrow \sum_{i=jB_h+1}^{\min[(j+1)B_h,n]}\bar{\mathbf{S}}, j \in (0, \lfloor\frac{l}{B_h}\rfloor), \mathbf{S}_b \in \mathbb{R}^{1\times\lceil\frac{l}{B_h}\rceil}$ ▷ Sum over blocks
Stage 2: Block selection
$(\mathbf{S}_b, \mathbf{I}) \leftarrow \text{sort}(\mathbf{S}_b)$
$\bar{\mathbf{I}} \leftarrow \{\mathbf{I}[:t] | \sum \mathbf{S}_b[:,:t] > \theta_1\}$ ▷ block indices
Stage 3: Sparse attention
$\mathbf{O} \leftarrow \text{Softmax}[\frac{\mathbf{q}\mathbf{k}[i,:,:]^T}{\sqrt{d^N+d^R}}]\mathbf{v}_t^C[i,:,:], i \in \bar{\mathbf{I}}$
return $\mathbf{O}$

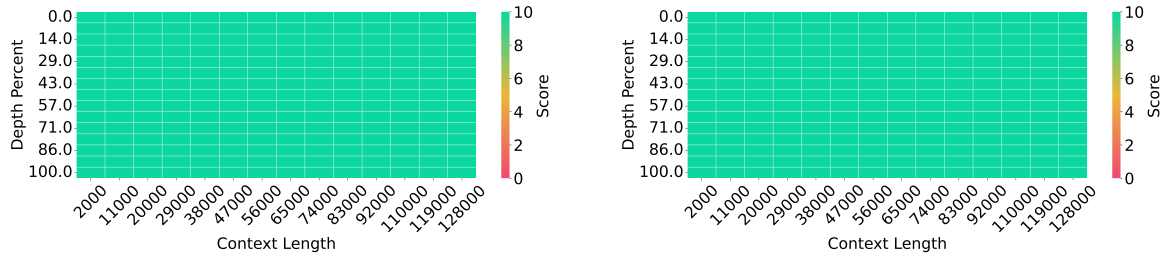
(MoE) model with 671B total parameters, of which 37B are activated for each token. DeepSeek-R1, derived from DeepSeek-V3, is a prominent reasoning model that utilizes a reinforcement learning framework to enhance its multi-step reasoning and long-context understanding capabilities. Both models feature a 128k context window and hundreds of billions of parameters. This choice of large-scale models distinguishes our work from previous studies on sparse attention. Our goal is to demonstrate the efficiency and effectiveness of RoBSA in real-world scenarios where large models processing long-context inputs face high latency and low throughput. This undertaking presents significant experimental challenges, as models of this scale are difficult to deploy, and their inference is time-consuming, particularly for DeepSeek-R1, which generates lengthy Chain-of-Thought (CoT) (Wei et al., 2023) outputs.

Setups All experiments are conducted on a two-node cluster, with each node equipped with eight NVIDIA H100 GPUs (80GB HBM), connected via NVLink and NVSwitch. Our implementation is based on vLLM (Kwon et al., 2023), which manages the KV-Cache in a paged manner inspired by virtual memory techniques in operating systems (Kilburn et al., 2009). We deploy the models using a distributed inference setup with a tensor parallelism of 8 and a pipeline parallelism of 2. For the RoBSA hyperparameters, we evaluate three combinations: $(\theta_1 = 0.95, \theta_2 = 0.99)$, $(\theta_1 = 0.9, \theta_2 = 0.99)$, and $(\theta_1 = 0.9, \theta_2 = 0)$. The setting $\theta_2 = 0$ indicates that all layers use only $\mathbf{q}^R$ and $\mathbf{k}^R$ to compute attention scores. We set the block size to $B_h = 16$, a choice informed by our observations in the previous section and a heuristic analysis. A detailed discussion of these hyperparameter settings is provided in the ablation study. We use standard MLA as the primary baseline in our main experiments, since RoBSA targets inference-time acceleration for large-scale MLA models, and there are currently no directly comparable sparse-attention methods that can be applied in this setting without additional training. We discuss DSA separately in Appendix C, as it requires substantial continued pre-training.

3.2 Performance Comparison

Needle-in-a-Haystack (NIAH) The NIAH benchmark (Kamradt, 2024) evaluates a model's ability to retrieve specific information from a large volume of text. As shown in Figure 7, RoBSA performs on par with full attention on the DeepSeek-V3 model across all evaluated sequence lengths and context depths, effectively preserving the model's retrieval capabilities. The corresponding results for DeepSeek-R1 are provided in Figure 8.

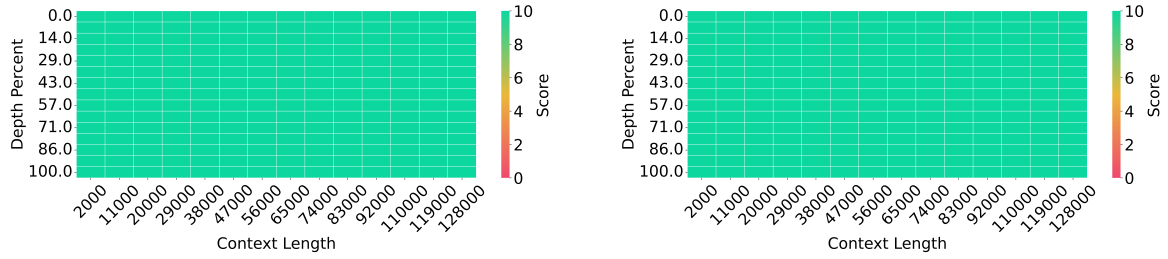
LongBench v2 LongBench v2 (Bai et al., 2025) is a diverse, large-scale benchmark for evaluating long-context reasoning which contains prompts with context length ranging from 8k to 2M words, with the majority under 128k. For prompts exceeding the model's 128k context window, we truncate them from the middle. Moreover, the maximum output length for DeepSeek-R1 was set to 8192 to prevent premature truncation. As shown in Table 1, RoBSA with $(\theta_1 = 0.9, \theta_2 = 0)$ exhibits a significant performance degradation for both DeepSeek-V3 and DeepSeek-R1, with the latter showing a more pronounced drop of approximately 5 points. In contrast, the $(\theta_1 = 0.9, \theta_2 = 0.99)$ setting slightly outperforms full attention on short-context



(a) Pressure test for Full MLA

(b) Pressure test for RoBSA

Figure 7: RoBSA retains the powerful retrieval capabilities of full MLA in the NIAH pressure test.



(a) Pressure test for Full MLA

(b) Pressure test for RoBSA

Figure 8: NIAH results for DeepSeek-R1.

subtasks but underperforms on medium and long ones, which may suggest that this configuration loses some information in longer contexts.

InfiniteBench InfiniteBench (Zhang et al., 2024) evaluates models on extremely long contexts (100k+ tokens). The results are presented in Table 2. We observe that RoBSA with $(\theta_1 = 0.9, \theta_2 = 0)$ undergoes a severe performance drop on *Retrieve_KV* and *Code_run* tasks; we provide a qualitative analysis of these failures in Appendix D. However, the $(\theta_1 = 0.9, \theta_2 = 0.99)$ configuration shows excellent performance, with minimal degradation overall and even outperforming full attention on specific subtasks like *Code_debug*. On tasks where full attention achieves perfect scores (e.g., *Retrieve_passkey*, *Retrieve_Numer*), RoBSA maintains this capability without any performance loss.

LV-Eval The LV-Eval benchmark (Yuan et al., 2024) is designed to rigorously assess model capabilities across varying input lengths. Detailed results comparing full MLA and RoBSA are shown in Table 3. We observe that RoBSA maintains or even slightly improves upon the performance of full attention on several subtasks. Despite minor decreases in some areas, the average scores indicate that RoBSA's overall performance is at least on par with the standard model. We also evaluated the $(\theta_1 = 0.9, \theta_2 = 0)$ configuration, which resulted in a notable performance decline; these results are presented in our ablation study (Table 5) to analyze the impact of layer partition threshold θ_2.

Conclusion We conducted comprehensive accuracy evaluations on several long-context benchmarks. Our results demonstrate that RoBSA, with appropriate hyperparameter settings (e.g., $(\theta_1 = 0.9, \theta_2 = 0.99)$), largely preserves the performance of the original full-attention model. However, the $(\theta_1 = 0.9, \theta_2 = 0)$ configuration leads to a significant performance drop on several tasks. We explore the impact of these hyperparameter choices in greater detail in the ablation study.

3.3 Efficiency Analysis

In this section, we analyze the efficiency improvements introduced by RoBSA. Given that a significant reduction in computation is meaningless if it causes a substantial loss in performance, we conduct our analysis with $\theta_2 = 0.99$, a setting demonstrated in the previous section to preserve model accuracy. RoBSA enhances decoding efficiency by substantially reducing the number of tokens involved in the attention computation.

Attention Computation Speedup The primary advantage of RoBSA is the reduction in attention computation, the main performance bottleneck for LLMs processing long contexts. Although the token selection algorithm introduces some overhead,

Model name	Attention type (θ_1, θ_2)	Easy	Hard	Short 0–32k	Medium 32k–128k	Long >128k	Overall
DeepSeek-V3	Full	**53.7**	44.3	50.8	**44.1**	**50.7**	47.9
	$(0.95, 0.99)$	53.6	43.1	50.0	43.7	49.1	47.1
	$(0.9, 0.99)$	53.0	**45.2**	**53.1**	43.6	49.1	**48.2**
	$(0.9, 0)$	47.9	38.9	43.9	39.5	45.4	42.3
DeepSeek-R1	Full	**65.6**	**55.6**	**62.8**	**58.6**	55.6	**59.4**
	$(0.95, 0.99)$	64.1	51.8	61.1	53.0	55.6	56.5
	$(0.9, 0.99)$	64.1	51.8	60.6	53.0	**56.5**	56.5
	$(0.9, 0)$	60.4	50.2	60.6	47.9	55.6	54.1

Table 1: Performance comparison on LongBench v2. RoBSA achieves competitive results with specific hyperparameter settings, but performance degrades substantially in other scenarios.

Model name	Attention type (θ_1, θ_2)	Retrieve_passkey 122.4k	Retrieve_Numer 122.4k	Retrieve_KV 89.9k	En_dia 103.6k	En_sum 171.5k	En_mc 184.4k	En_qa 192.6k	Zh_qa 2068.6k	Code_run 75.2k	Code_debug 114.7k	Avg -
DeepSeek-V3	Full	100	**100**	**96.8**	**34.00**	**27.64**	80.26	**23.54**	**25.52**	**42.25**	25.89	**55.59**
	$(0.9, 0.99)$	**100**	100	94	31.00	26.67	**81.14**	21.58	25.32	38.50	**26.96**	54.52
	$(0.9, 0)$	100	99.83	6.6	31.54	26.75	80.64	21.46	25.64	12	24.11	42.86

Table 2: Evaluation on the comprehensive tasks in InfiniteBench. Scores in red denote a severe performance drop compared to the baseline.

Dataset name	16k	32k	64k	128k	Avg
cmrc_mixup	41.47 $_{+1.28}$	39.85 $_{-0.41}$	37.22 $_{-0.32}$	37.59 $_{-0.32}$	39.03$_{+0.05}$
dureader_mixup	24.55 $_{+1.40}$	22.58 $_{-0.24}$	22.18 $_{+1.50}$	22.17 $_{-1.41}$	22.87$_{+0.31}$
factrecall_en	39.97 $_{+1.20}$	36.85 $_{+0.22}$	34.52 $_{+0.42}$	34.19 $_{-0.39}$	36.38$_{+0.36}$
hotpotwikiqa_mixup	62.09 $_{+1.70}$	62.40 $_{-0.07}$	60.97 $_{+0.55}$	58.84 $_{-0.40}$	61.07$_{+0.44}$
lic_mixup	50.09 $_{-0.27}$	47.86 $_{-1.01}$	46.39 $_{+0.31}$	39.11 $_{+0.27}$	45.86$_{+0.33}$

Table 3: Comparison of Full Attention and RoBSA ($\theta_1 = 0.9, \theta_2 = 0.99$) on LV-Eval for DeepSeek-V3. Each value represents the score for RoBSA, with the subscript indicating the performance change relative to the baseline. Changes are color-coded: green for improvement and red for decrease.

it is essential to ensure that the overall speedup remains substantial enough to compensate for this cost. Figure 9 compares the attention kernel latency between standard MLA and RoBSA. As observed, the speedup becomes significant for long contexts, achieving over a 10x reduction in latency for context lengths exceeding 32k. This gain is driven by a dramatic reduction in the number of tokens processed, as shown in Figure 10. For a context length of 128k, the number of blocks (with a block size of 16) is reduced by a factor of up to 16, down to approximately 500 blocks (equivalent to 8,000 tokens). This substantial reduction in computational load is the primary driver of the observed kernel latency improvements.

End-to-End Speedup While RoBSA significantly accelerates the attention kernel, it is crucial to quantify its benefit in a complete, end-to-end model serving scenario. To this end, we evaluate the end-to-end latency using **Time-Per-Output-Token (TPOT)**, which measures the average time to generate each consecutive output token. Unlike kernel-level metrics, TPOT captures the performance experienced by end-users. As plotted in Figure 11, RoBSA achieves a significant reduction in TPOT for long-context settings, with greater speedups observed as the context length increases. With $\theta_1 = 0.9$ and a 128k context length, RoBSA achieves a $2.55\times$ speedup, reducing TPOT from approximately 175 ms to 60 ms. With $\theta_1 = 0.95$, the improvement is still substantial, yielding speedups of $2.03\times$ at 112k context and $2.15\times$ at 128k.

Furthermore, we evaluate RoBSA in a more realistic multi-batch serving scenario where multiple clients send requests concurrently. As shown in Figure 12, RoBSA demonstrates strong performance in this setting, increasing throughput by up to $1.79\times$ and reducing TPOT by a factor of $1.77\times$ with a batch size of 8. This result further demonstrates RoBSA's ability to efficiently accelerate long-context processing in practical applications.

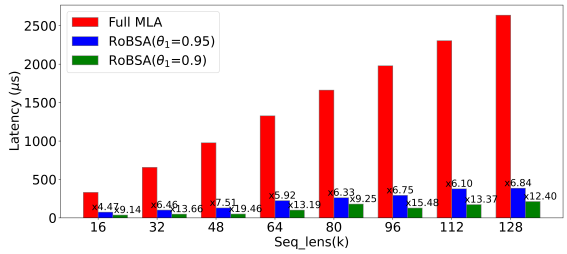


Figure 9: Latency comparison of the attention kernel for full MLA vs. RoBSA, demonstrating a substantial speedup for long sequences.

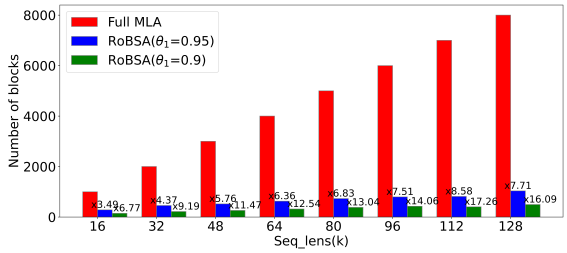


Figure 10: The number of active blocks in the attention computation for RoBSA vs. the full context length. The dramatic reduction in tokens is the primary reason for the kernel speedup.

4 Conclusion

In this paper, we first introduce the high latency problem under long-context situations for large language models with MLA where the bottleneck is the attention computation. Then we observe the sparse phenomenon in MLA and more interestingly, the high correlation between the attention score computed by full query-key pair and the partial query-key pair with RoPE. Based on this finding, we propose a lightweight sparse attention method called RoBSA for MLA, which utilizes the RoPE part of qk to compute attention score. To avoid random memory access, we implement blockwise token selection. Moreover, to better maintain model performance, we partition the layers into RoPE-sensitive and RoPE-insensitive ones considering the similarity of RoPE attention score and standard attention score. The result shows that RoBSA substantially improves throughput and latency in long-context scenarios while maintaining model ability. This paves the way to solve the long-context inference problems for large models especially the ones implemented with MLA.

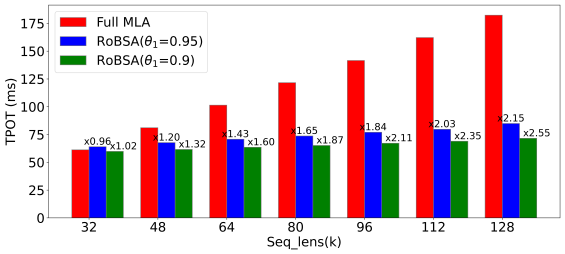


Figure 11: Time-Per-Output-Token (TPOT) comparison between full MLA and RoBSA with $\theta_2 = 0.99$. RoBSA significantly reduces latency, especially for longer contexts.

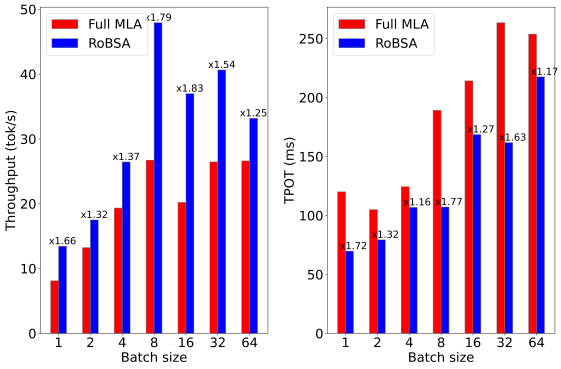


Figure 12: End-to-end speedup in a multi-batch setting. The x-axis represents the number of concurrent requests (batch size), with request lengths ranging from 16k to 128k.

Limitations

RoBSA only tackles the high latency problem in decoding stage during model inference and does not touch the prefilling stage where the computation cost is quadratic instead of linear with respect to context length. Moreover, RoBSA only applies in models with MLA structure which takes up only a small fraction of open-source and closed-source models, mainly the DeepSeek family and does not work for the mainstream MHA and GQA. Furthermore, the selection of hyperparameters remains a black box and seems more like a heuristic sense.

Acknowledgments

This work is supported by the National Key R&D Program of China (NO. 2025YFB3003704) and National Natural Science Foundation of China (Grant No. 62495062). We would also like to express our gratitude to Zhipu AI for providing the computing resources that supported our experiments.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *Preprint*, arXiv:2305.13245.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, and 3 others. 2025. Smollm2: When smol goes big – data-centric training of a small language model. *Preprint*, arXiv:2502.02737.

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. 2025. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *Preprint*, arXiv:2412.15204.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *Preprint*, arXiv:2004.05150.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. 2024. Magicpig: Lsh sampling for efficient llm generation. *Preprint*, arXiv:2410.16179.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *Preprint*, arXiv:1904.10509.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 3290 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261.

Steve Dai, Hasan Genc, Rangharajan Venkatesan, and Brucek Khailany. 2023. Efficient transformer inference with statically structured sparse attention. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE.

DeepSeek-AI. 2025. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, and 138 others. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025b. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. *Preprint*, arXiv:2301.00774.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers. *Preprint*, arXiv:2210.17323.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive kv cache compression for llms. *Preprint*, arXiv:2310.01801.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

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

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. 2024. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Preprint*, arXiv:2407.02490.

Greg Kamradt. 2024. Llmtest_needleinahaystack: Doing simple retrieval from llm models at various context lengths to measure accuracy. https://github.com/gkamradt/LLMTest_NeedleInAHaystack. Accessed: 2024-05-23.

Tom Kilburn, David BG Edwards, Michael J Lanigan, and Frank H Sumner. 2009. One-level storage system. *IRE Transactions on Electronic Computers*, (2):223–235.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. *Preprint*, arXiv:2309.06180.

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, and 6 others. 2025. Moba: Mixture of block attention for long-context llms. *Preprint*, arXiv:2502.13189.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *Preprint*, arXiv:1911.02150.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding. *Preprint*, arXiv:2104.09864.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *Preprint*, arXiv:2406.10774.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, and 150 others. 2025. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Hanrui Wang, Zhekai Zhang, and Song Han. 2021. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024a. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Preprint*, arXiv:2402.04617.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024b. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *Preprint*, arXiv:2410.10819.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024c. Efficient streaming language models with attention sinks. *Preprint*, arXiv:2309.17453.

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, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. 2025b. Parallelizing linear transformers with the delta rule over sequence length. *Preprint*, arXiv:2406.06484.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, Vienna, Austria. Association for Computational Linguistics.

Tao Yuan, Xuefei Ning, Dong Zhou, Zhijie Yang, Shiyao Li, Minghui Zhuang, Zheyue Tan, Zhuyu Yao, Dahua Lin, Boxun Li, Guohao Dai, Shengen Yan, and Yu Wang. 2024. Lv-eval: A balanced long-context benchmark with 5 length levels up to 256k. *Preprint*, arXiv:2402.05136.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. 2025. Spargeattention: Accurate and training-free sparse attention accelerating any model inference. *Preprint*, arXiv:2502.18137.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024. ∞bench: Extending long context evaluation beyond 100k tokens. *Preprint*, arXiv:2402.13718.

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. 2023. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. *Preprint*, arXiv:2306.14048.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A survey on efficient inference for large language models. *Preprint*, arXiv:2404.14294.

A Related Work

Sparse Attention, in particular, has emerged as one of the most promising and widely adopted strategies. By restricting each token's receptive field to a subset of positions, sparse attention mechanisms reduce the computational complexity of attention from quadratic to sub-quadratic. Structured sparsity (e.g., local windows (Dai et al., 2023), block patterns (Beltagy et al., 2020)) and adaptive sparsity (Wang et al., 2021) (learned or dynamic token selection) allow models to capture long-range dependencies without incurring the cost of full attention. More specifically, StreamingLLM (Xiao et al., 2024c) adopts a local attention pattern while applying a global pattern only to the first few tokens. This global pattern acts as an *attention sink*, maintaining strong attention to the initial tokens, which in turn enables LLMs to generalize to infinitely long input sequences. Following this work, DuoAttention (Xiao et al., 2024b) further discovered that the *attention sink* phenomenon does not apply in all heads and layers. They come up with a method to partition the heads into *Streaming heads* which primarily focus on recent tokens and attention sinks and *retrieval heads*, which attend to all previous tokens. H2O (Zhang et al., 2023) introduces a token-level dynamic attention pruning mechanism. It integrates static local attention with dynamic computations between the current query and a set of dynamically identified key tokens, referred to as heavy-hitters (H2). The heavy-hitter set is updated with an eviction policy that removes the least significant keys at each generation step, thereby maintaining both the size and relevance of the set. FastGen (Ge et al., 2024) leverages adaptive KV cache compression to intelligently reduce memory usage during generative inference, achieving substantial efficiency gains without compromising output quality. Quest (Tang et al., 2024) introduces a query-aware KV cache selection strategy that dynamically identifies and loads only the most critical cache pages and significantly speeds up self-attention by only loading the Top-K critical KV cache pages for attention. InfLLM (Xiao et al., 2024a) stores distant contexts in auxiliary memory units and uses an efficient mechanism to retrieve token-relevant units for attention computation. MInference (Jiang et al., 2024) identifies three patterns in long-context attention: the *A-shape*, *Vertical-Slash*, and *Block-Sparse*. Based on their findings, they leverage pattern-aware sparse computation to accelerate pre-filling in long-context LLMs. SpargeAttention (Zhang et al., 2025) proposes a two-stage online filtering strategy for sparse computation. First, they compress each block of Q and K into a single representative token based on the similarity among tokens within each block, then they use it to construct a sparse mask which is utilized in the next stage to perform a sparse online softmax algorithm. MagicPIG (Chen et al., 2024) claims that the prevailing TopK attention itself is problematic because attention is not always as sparse as expected which leads to its sufferings from quality degradation in certain downstream tasks. They introduce a sampling algorithm for attention estimation based on locality sensitive hashing which is totally different from traditional topk selection in sparse attention. MoBA (Lu et al., 2025) embodies the "less structure" principle by enabling models to autonomously determine attention, seamlessly balancing full and sparse modes. NSA (Yuan et al., 2025) offers a natively trainable sparse attention mechanism that unifies algorithmic efficiency with hardware-aware optimization. By combining hierarchical token compression with fine-grained selection, it preserves global context while ensuring local precision.

B Ablation Studies

In this section, we study various aspects of RoBSA and evaluate the design choices we make with ablation experiments.

B.1 Impact of Layer Partition

We discuss why we choose to classify the layers into RoPE-sensitive and RoPE-insensitive ones and how we choose the hyperparameter θ_2. As shown in Table 5, when we treat all layers as RoPE-sensitive, RoBSA experiences very bad performance degradation with some subtasks dropping over 10 points. Compared to results in Table 3,

Model Name	DeepSeek-V3						DeepSeek-R1					
θ_2	0.95	0.96	0.97	0.98	0.99	0.995	0.95	0.96	0.97	0.98	0.99	0.995
Number of RoPE-insensitive Layers	11	12	17	22	31	38	10	13	16	20	30	39

Table 4: When $\theta_2 > 0.99$, more than half of the layers are considered RoPE-insensitive.

Dataset name	16k	32k	64k	128k	Avg
cmrc_mixup	35.34 $_{-4.85}$	32.19 **-8.07**	30.60 **-6.94**	31.31 **-6.60**	32.36 **-6.62**
dureader_mixup	19.22 $_{-3.93}$	17.99 $_{-4.83}$	19.07 $_{-1.61}$	17.54 **-6.04**	18.46 $_{-4.10}$
factrecall_en	8.01 $_{0.29}$	7.87 $_{-0.28}$	8.55 $_{0.10}$	11.79 $_{1.98}$	9.05 $_{0.52}$
hotpotwikiqa_mixup	40.83 $_{-2.26}$	43.29 $_{3.70}$	36.35 $_{0.07}$	24.13 $_{-1.62}$	36.15 $_{-0.03}$
lic_mixup	33.01 $_{-4.96}$	31.61 **-5.02**	24.93 **-10.01**	22.36 **-12.19**	27.98 **-8.54**

Table 5: Comparison of Full MLA and RoBSA($\theta_1 = 0.9, \theta_2 = 0$) on LV-eval of DeepSeek-V3. The number denotes the scores of RoBSA and are annotated with a subscript indicating the performance change relative to standard DeepSeek-V3. Performance changes are color-coded: bold green (improvement), red (decrease) and dark red(decrease > 5).

Dataset name	$\theta_1 = 0.9$	$\theta_1 = 0.95$	$\theta_1 = 0.98$	$\theta_1 = 0.995$	$\theta_1 = 1.0$**(full)**
Retrieve_KV	6.6	14.4	42.4	82.6	94.68
Code_run	12	23.5	28.25	40.0	42.25

Table 6: Performance increases with bigger θ_1 where more token blocks are taken into computation (here we fix $\theta_2 = 0$).

where only θ_2 changes, we observe the effectiveness of our layer partition method. Here we also provide the number of RoPE-insensitive layers with respect to the change of θ_2 in Table 4. $\theta_2 = 0.99$ is a preferable choice where the model proves to maintain its capability in all tasks and the number of RoPE-insensitive layers are proper enough.

B.2 Selection of Calibration Dataset

In Section 2.3.1 we only demonstrate the selection of the calibration dataset for layer partition but not mention how we choose it. In this section, we illustrate that the choice of calibration data can influence the resulting layer partition. As shown in Table 7. Both the domain characteristics and the context length of the calibration set affect which layers are identified as RoPE-insensitive. For example, short prompts (e.g., MMLU, <1k tokens) tend to identify fewer RoPE-insensitive layers, whereas long prompts (e.g., RULER, ∼15k tokens) expose more layers as insensitive. Concretely, using 20 prompts from MMLU identifies 19 layers as RoPE-insensitive, while 20 prompts from RULER identifies a substantially larger set of 32 layers. To mitigate this bias, we construct a mixed calibration dataset combining 100 prompts from diverse domains and lengths (SmolTalk, MMLU, HumanEval, GSM8K, and RULER), whose detailed composition is provided in Table 8, so that the resulting partition reflects the model's intrinsic behavior rather than being biased towards any single input distribution. Our experiments on NIAH, LongBench v2, InfiniteBench, and LV-Eval show that this static partition generalizes well across tasks from different domains, confirming that layer-wise sensitivity is primarily a model-intrinsic property.

B.3 Impact of Block Selection Threshold θ_1

We also look at the impact of threshold θ_1 in the token selection part. From previous sections, we already get a glimpse of the impact of θ_1. In efficiency analysis, we see that bigger θ_1 brings more computation overhead because they take more tokens into account. In accuracy experiments, it seems that the change of θ_1 does not affect performance that much. For example, in Longbench v2, with $\theta_2 = 0.99$, change of θ_1 from 0.9 to 0.95 only add one to overall score for DeepSeek-V3, and does not change that of DeepSeek-R1. For the two subtasks which experience severe drop in InfiniteBench in Table 2, we try to vary θ_1 to see how it affects the performance. As shown in Table 6, when θ_1 increases from 0.9 to 0.99, the score on Retrieve_KV increase from 6.6 to 82.6 and Code_run from 12 to 40.0. This shows that bringing much information to compute is beneficial even though

Calibration Set	#Samples	Context Length	Identified RoPE-Insensitive Layers	#Layers
MMLU	20	<1k	24, 26, 27, 28, 29, 30, 31, 32, 33, 36, 38, 39, 40, 43, 46, 47, 57, 58, 60	19
RULER	20	≈15k	4, 17, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 45, 46, 47, 50, 55, 60	32

Table 7: Layer partition results for DeepSeek-V3 under different calibration datasets

Source	Task Type	#Samples
SmolTalk	Dialogue	10
MMLU	Knowledge QA	20
HumanEval	Code generation	20
GSM8K	Math reasoning	20
RULER	Long-context	30
Total	Mixed	100

Table 8: Composition of the mixed calibration dataset for layer sensitivity analysis

there is redundancy sometimes.

Seq_len (k)	$B_h = 16$	$B_h = 32$	$B_h = 64$	$B_h = 128$
32	59	60	62	63
48	61	62	64	65
64	64	65	66	70
80	65	67	68	73
96	67	69	71	75
112	69	71	72	76
128	72	73	75	78

Table 9: Time-Per-Output-Token (ms) for different block sizes and sequence lengths.

B.4 Impact of Block Size

Block size B_h controls the granularity of sparsification: a smaller block enables finer-grained selection but increases scheduling overhead, while a larger block reduces overhead but retains more tokens per selected block. Table 12 reports LongBench v2 accuracy for different block sizes with $(\theta_1 = 0.9, \theta_2 = 0.99)$ on DeepSeek-V3.

Accuracy remains stable across all block sizes, demonstrating strong robustness of RoBSA to this hyperparameter. However, block size does affect decoding latency (TPOT). Table 9 shows Time-Per-Output-Token (ms) for different sequence lengths.

To explain the latency trend, Tables 10 and 11 report the number of active blocks and active tokens under each configuration.

As block size grows, the number of active blocks decreases but the total number of active tokens increases. This is because larger blocks reduce selection granularity: once a block is selected, all tokens within it must be retained. For instance, at a 128k context, $B_h = 128$ selects only 100 blocks but retains 12,800 active tokens, whereas $B_h = 16$ selects 500 blocks with only 8,000 active tokens. Since attention computation cost scales with the number of active tokens rather than the number of blocks, a larger block size directly increases the computation load and therefore raises TPOT. This analysis motivates choosing a smaller block size ($B_h = 16$ or $B_h = 32$) to minimize latency while preserving accuracy.

Seq_len (k)	$B_h = 16$	$B_h = 32$	$B_h = 64$	$B_h = 128$
32	220	130	65	35
48	260	150	90	50
64	320	180	110	65
80	380	220	130	75
96	430	250	150	80
112	450	260	160	85
128	500	320	180	100

Table 10: Number of active blocks selected by RoBSA for different block sizes and sequence lengths.

Seq_len (k)	$B_h = 16$	$B_h = 32$	$B_h = 64$	$B_h = 128$
32	3520	4160	4160	4480
48	4160	4800	5760	6400
64	5120	5760	7040	8320
80	6080	7040	8320	9600
96	6880	8000	9600	10240
112	7200	8320	10240	10880
128	8000	10240	11520	12800

Table 11: Number of active tokens involved in attention computation for different block sizes and sequence lengths.

C Comparison with DSA

DeepSeek Sparse Attention (DSA) (DeepSeek-AI, 2025) is currently the only other sparse attention mechanism designed specifically for MLA, mak-

Block Size	Easy	Hard	Short	Medium	Long	Overall
Full Attention	52.1	45.0	51.1	45.1	47.2	47.7
$B_h = 16$	53.6	43.1	50.0	43.7	49.1	47.1
$B_h = 32$	52.6	45.0	52.8	43.7	48.1	47.9
$B_h = 64$	52.6	44.7	50.6	44.2	50.0	47.7
$B_h = 128$	52.6	45.3	52.8	43.3	50.0	48.1

Table 12: LongBench v2 accuracy under different block sizes. RoBSA performance is consistent across all evaluated block sizes.

Model	Easy	Hard	Short	Medium	Long	Overall
DeepSeek-V3.1-Terminus	58.3	**46.6**	**58.9**	47.0	46.3	51.1
DeepSeek-V3.2-Exp (DSA)	54.2	46.6	54.4	47.0	46.3	49.5
V3.1-Terminus + RoBSA	**60.9**	46.0	57.2	**48.8**	**48.1**	**51.7**

Table 13: Comparison of RoBSA and DSA on LongBench v2(DeepSeek-V3.2-Exp = DeepSeek-V3.1-Terminus + DSA). RoBSA applied to DeepSeek-V3.1-Terminus achieves higher overall accuracy than DSA while requiring no additional training.

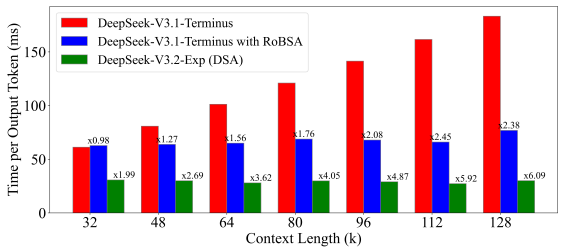


Figure 13: Time-Per-Output-Token (TPOT) comparison between DSA and RoBSA.

ing it the most direct baseline. DSA employs a *Lightning Indexer* followed by *fine-grained top-k token selection* for the full attention computation. Crucially, DSA requires two stages of continued pre-training on DeepSeek-V3.1-Terminus: a dense warm-up stage ($\sim$2.1B tokens) and a sparse training stage ($\sim$943.7B tokens), totaling $\sim$945.8B tokens. RoBSA, by contrast, is entirely training-free.

Accuracy. Table 13 compares the two methods on LongBench v2. RoBSA achieves an overall score of 51.7, surpassing both DSA (49.5, -1.6 vs. baseline) and even the full-attention baseline (51.1). Notably, DSA incurs a regression on Easy tasks (54.2 vs. 58.3 baseline, -4.1), whereas RoBSA achieves a gain (60.9, $+2.6$). On Medium and Long subsets, RoBSA exceeds the baseline by $+1.8$ points each, while DSA matches it exactly.

Efficiency. We further compare end-to-end decoding latency (TPOT) on DeepSeek-V3.1-Terminus. As shown in Figure 13, DSA achieves a $6.09\times$ speedup over full MLA (TPOT $\approx$30 ms) under 128k context by internalizing an aggressive sparsity ratio through large-scale re-training. RoBSA achieves a $2.38\times$ speedup (TPOT $\approx$75 ms) without any training. The efficiency gap reflects a fundamental trade-off: DSA can adopt a more aggressive sparsity budget because its accuracy cost is absorbed during training, while RoBSA's threshold θ_1 is constrained by the need to preserve accuracy without supervision. In summary, DSA holds a clear efficiency advantage, but RoBSA surpasses DSA in accuracy on LongBench v2 without incurring $\sim$945.8B tokens of re-training, making it a compelling plug-and-play alternative for deployments where training resources are limited.

D Example Prompts of Full MLA and RoBSA

In this section, we present several qualitative examples observed when applying RoBSA to DeepSeek-V3. We choose one prompt from the Retrieve_KV and Code_run subtask from InfiniteBench respectively where RoBSA undergoes a bad performance degradation mentioned in Table 2. We compare the output of standard DeepSeek-V3 and DeepSeek-V3 with RoBSA($\theta_1 = 0.9, \theta_2 = 0$). The prompts are truncated here due to page limits. We highlight the positions where RoBSA makes errors in yellow box .

D.1 Retrieve_KV

Retrieve_KV

Prompt

Extract the value corresponding to the specified key in the JSON object below.
JSON data:

{"798c2306-5ad1-42a9-a8de-f2a118b33744": "5e6b7b90-710d-4953-9b18-3e96b2cadbf2",
"6ab6ea3e-f288-4f33-ba46-7f42bb75b03f": "cb59052b-9128-4979-9c0e-e1de4adcf73b",
"7f8461f4-93c2-4536-9cfa-5ce11560fa07": "3b2dd49b-7437-4432-ac24-06fb8f7f862e",
"d10cfc9c-364b-4b62-b522-5effbd5b41b0": "8c91d8da-9bd2-4a95-931e-15aa06eb7056",
"2034ca7d-7546-41b1-849a-e19262a4765d": "05d0befa-7662-4c00-a2d4-d80a2068781c",
"3f41ebd0-7e41-41f9-af56-75ec8920b5ac": "6290fbbe-66ce-4730-b038-6bd82ede18b3",
"7d7b5c42-46ea-45ca-999c-5ec77cc04ffc": "db26cfee-7278-4a60-987e-5826cf2bf04d",
"905ea369-f5c4-4ad1-a94f-0b1610e09268": "4f44174e-ae38-4080-9122-26f388a2d697"

. . . . . .

"6f734370-579d-45f4-95a6-c3b67cc35fd9": "abcc1ad3-3d76-45d8-aa59-a4d9f7a9d46a",
"fdf69034-1fbd-42d1-b3b6-620fb37278f8": "8a365300-5150-4e4f-9629-4a73c8d03424",
"8fde9aa2-73d8-465d-8678-7da878bf8812": "77cf5917-b238-44a7-840a-a81fa8f4eeec",
"69b8ea2d-da43-4f87-8856-741b1a9f8450": "1f5eba0d-5ccf-4262-aa76-d7fbabdc0b9a"}

Key: "842f5dfb-448f-438f-881a-203283f8cdd1"
The value associated with the specified key is:

Gold Answer

bbf3fa73-6217-4e6e-923c-8349bd286c3d

Output for Standard DeepSeek-V3

The value associated with the key "842f5dfb-448f-438f-881a-203283f8cdd1" is:

"bbf3fa73-6217-4e6e-923c-8349bd286c3d"

Output for RoBSA($\theta_1 = 0.9, \theta_2 = 0$)

The value associated with the key "842f5dfb-448f-438f-881a-203283f8cdd1" is:

"bbf3fa73-6217-4e6e-923c-8349bd286 e 3d"

Code_run

Prompt

Following is a set of Python functions. There is a function called named func_1069. Here is the content of functions_module.py:

def func_1(x): return func_8226(x) - 1

def func_2(x): return func_6757(x)

. . . . . .

def func_1069(x): return func_6456(x) - 1

. . . . . .

def func_6456(x): return x + 4

. . . . . .

def func_8798(x): return func_6676(x) + 8

Please give me the exact number of the return value of func_1069(5). Be concise. Your response must end with the final returned value.

Gold Answer

8

Output for Standard DeepSeek-V3

The function func_1069(x) calls func_6456(x) - 1. The function func_6456(x) returns x + 4. For x = 5, func_6456(5) returns 5 + 4 = 9. Therefore, func_1069(5) returns 9 - 1 = 8. The final returned value is 8.

Output for RoBSA($\theta_1 = 0.9, \theta_2 = 0$)

To find the return value of func_1069(5), we need to follow the function calls:

1. func_1069(x) calls func_6456(x) and subtracts 1.

2. func_6456(x) is not defined in the provided code , so we assume it returns x directly.

Thus, func_1069(5) = func_6456(5) - 1 = 5 - 1 = 4.

The final returned value is 4.

We can see that RoBSA makes mistakes by either retrieving incorrect information or losing important information.