TokenSelect: Efficient Long-Context Inference and Length Extrapolation for LLMs via Dynamic Token-Level KV Cache Selection

Wei Wu[1†‡], **Zhuoshi Pan**[2†‡], **Kun Fu**[3], **Chao Wang**[1], **Liyi Chen**[4‡],
Yunchu Bai[1], **Tianfu Wang**[5], **Zheng Wang**[3], **Hui Xiong**[5,6*]

[1]University of Science and Technology of China, [2]Tsinghua University,
[3]Alibaba Cloud Computing, [4]Xiaohongshu Inc.,
[5]The Hong Kong University of Science and Technology (Guangzhou),
[6]The Hong Kong University of Science and Technology

urara@mail.ustc.edu.cn, xionghui@ust.hk

Abstract

Rapid advances in Large Language Models (LLMs) have spurred demand for processing extended context sequences in contemporary applications. However, this progress faces two challenges: performance degradation due to sequence lengths out-of-distribution, and excessively long inference times caused by the quadratic computational complexity of attention. These issues limit LLMs in long-context scenarios. In this paper, we propose Dynamic Token-Level KV Cache Selection (*TokenSelect*), a training-free method for efficient and accurate long-context inference. *TokenSelect* builds upon the observation of non-contiguous attention sparsity, using QK dot products to measure per-head KV Cache criticality at token-level. By per-head soft voting mechanism, *TokenSelect* selectively involves a few critical KV cache tokens in attention calculation without sacrificing accuracy. To further accelerate *TokenSelect*, we design the Selection Cache based on observations of consecutive Query similarity and implemented the efficient Paged Dot Product Kernel, significantly reducing the selection overhead. A comprehensive evaluation of *TokenSelect* demonstrates up to 23.84× speedup in attention computation and up to 2.28× acceleration in end-to-end latency, while providing superior performance compared to state-of-the-art long-context inference methods.

1 Introduction

With the rapid development of large language models (LLMs), the number of parameters is no longer the sole factor significantly affecting model performance. The ability to effectively process longer context information has become one of the key metrics for evaluating LLMs' capabilities. Recent advances—such as cross-document understanding (Bai et al., 2024), LLM-powered search systems (Sharma et al., 2024), complex reasoning (OpenAI), and other cutting-edge LLM developments (Wu et al., 2025a; Pan et al., 2025; Wu et al., 2025b; Chen et al., 2024)—have all placed higher demands on the long-context capabilities of LLMs. There are two main difficulties in using pre-trained LLMs for long-context inference. On one hand, LLMs are limited by their context length during pre-training (*e.g.* Llama 3 only has 8192 tokens). Directly inferencing on longer sequences can lead to severe performance degradation due to reasons including sequence lengths out-of-distribution (Xiao et al., 2024b; Han et al., 2024). On the other hand, even if LLMs possess sufficiently large context lengths, the quadratic computational complexity of attention with respect to sequence length makes the response time for long-context inference unbearable.

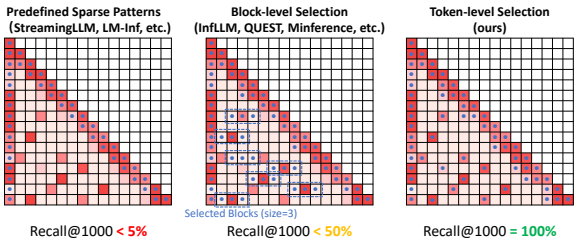


Figure 1: Distribution of tokens participating in attention computation under different sparsity patterns (blue dots. *TokenSelect* can more accurately select critical tokens (crimson squares) for attention computation.

Previous works have made numerous attempts to address these difficulties. To extend the context length of LLMs, the current common practice is to perform post-training on long texts (Yang et al., 2024a). However, this approach entails significant computational costs, motivating a training-free and effective method that is computationally efficient.

*Corresponding Author.
†Equal Contribution.
‡Work during internship at Alibaba Cloud Computing.
§Code: https://github.com/pzs19/TokenSelect

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 21276–21293
November 4-9, 2025

To accelerate long-context inference, many studies focus on the sparsity of attention, attempting to reduce the scale of KV Cache involved in computation. The key to this type of method lies in designing sparse patterns for attention, which can be mainly divided into two categories: one uses predefined sparse patterns (Wang et al., 2019; Zaheer et al., 2020; Xiao et al., 2024b; Han et al., 2024), while the other estimates the potential importance of KV Cache during the inference process (Zhang et al., 2024b; Li et al., 2024; Lee et al., 2024; Tang et al., 2024; Jiang et al., 2024; Sun et al., 2025), attempting to select relevant KV Cache tokens into attention calculations. However, the design of these sparse patterns is often heuristically based on historical criticality or coarse-grained criticality estimation of tokens, making it difficult to ensure that the selected tokens are truly critical, thus resulting in sub-optimal performance, as shown in *Fig.* 1.

In this paper, we further observe the non-contiguous sparsity of attention, revealing the importance of designing more fine-grained dynamic sparse patterns. To this end, we propose *TokenSelect*, a training-free approach that utilizes token-level selective sparse attention for efficient long-context inference and length extrapolation. Specifically, for each Query, *TokenSelect* dynamically calculates token-level per-head criticality for the past KV Cache and selects the k most critical tokens through our head soft vote mechanism, involving them in the attention calculation. This reduces the scale of attention calculation to a constant length familiar to the model, while maintaining almost all of the long-context information, thereby simultaneously addressing the two main difficulties for long-context inference. To reduce the overhead of token selection, *TokenSelect* manages the KV Cache in token-level pages (Zheng et al., 2024) and design efficient kernel for token selection based on paged KV Cache management through Triton (Tillet et al., 2019). Furthermore, based on our observation of high similarity between consecutive queries, we have designed the Selection Cache, which allows consecutive similar queries to share token selection results, thereby reducing the selection frequency while ensuring its effectiveness.

We evaluate the performance and efficiency of *TokenSelect* on three representative long-context benchmarks using three open-source LLMs. The experimental results demonstrate that our *TokenSelect* can achieve up to $23.84\times$ speedup in attention computation compared to FlashInfer (flashinfer ai), and up to $2.28\times$ acceleration in end-to-end inference latency compared to state-of-the-art long-context inference method (Xiao et al., 2024a). Simultaneously, it provides superior performance on three long-text benchmarks. In summary, we make the following contributions:

- An observation on the non-contiguous sparsity of attention that highlights the importance of token-level KV Cache selection.
- *TokenSelect*, a training-free method that achieves accurate and efficient long-context inference and length extrapolation, which is compatible with mainstream LLM serving systems.
- Comprehensive evaluations of our method, showing up to $23.84\times$ speedup in attention computation and up to $2.28\times$ acceleration in end-to-end latency while exhibiting superior performance.

2 Related Works

Long-context LLMs. Due to computational complexity constraints, current LLMs based on Transformers often utilize limited context lengths during pre-training (Touvron et al., 2023; Dubey et al., 2024; Jiang et al., 2023; Yang et al., 2024a; GLM et al., 2024; AI et al., 2024). To extend their long-context capabilities, existing methods can be broadly categorized into three approaches (Huang et al., 2024; Zhou et al., 2024; Zhao et al., 2023): 1) Modifying positional encodings: A widely adopted method is positional interpolation (Chen et al., 2023). Chen et al. first proposed linear scaling of RoPE (Su et al., 2024) to map longer positional ranges within the original training window. Subsequent works (bloc97, 2023; emozilla, 2023) further improved this method using Neural Tangent Kernel (NTK) theory (Jacot et al., 2018), achieving longer context windows while maintaining model performance. Methods like YaRN (Peng et al., 2024) and Giraffe (Pal et al., 2023) optimize interpolation effects by adjusting frequency components or introducing temperature parameters. 2) Long-context post-training: This approach extends the model's context length through additional training steps on longer documents after pre-training (Yang et al., 2024b; Tian et al., 2025). It has been widely adopted by leading LLMs (Team et al., 2024; Yang et al., 2024a; GLM et al., 2024) with the support of sequence parallelism techniques (Shoeybi et al., 2020; Jacobs et al., 2024; Liu et al., 2024b). 3) Incorporating additional memory modules: Notable examples

include Transformer-XL (Dai* et al., 2019), Compressive Transformer (Rae et al., 2020), RMT (Bulatov et al., 2022) and Infini-attention (Munkhdalai et al., 2024). Although these methods have expanded the context length of LLMs, long-context inference still faces the challenge of high computational costs.

Efficient Long-context Inference. In state-of-the-art LLMs serving systems (Kwon et al., 2023; Zheng et al., 2024), technologies such as Flash Attention (Dao, 2024) and Paged Attention (Kwon et al., 2023) have greatly optimized LLMs inference efficiency by improving GPU I/O bottlenecks. However, in long-context inference scenarios, the computational complexity of attention poses new challenges for LLMs inference. Numerous studies focus on the sparsity of attention, selecting partial KV Cache for attention calculations to improve long-context inference efficiency. Sliding window (Wang et al., 2019; Zaheer et al., 2020) is one of the most widely used sparse patterns, reducing complexity to linear by executing attention computations within localized windows. Recent works like StreamingLLM (Xiao et al., 2024b) and LM-infinite (Han et al., 2024) retain the initial tokens of the sequence in addition to sliding windows, effectively maintaining LLMs' performance when processing long sequences. While these approaches are simple to implement, they cannot retain information from long contexts. Another approach focuses on KV Cache eviction during inference. Methods like H_2O (Zhang et al., 2024b), TOVA (Oren et al., 2024) and SnapKV (Li et al., 2024) evaluate token criticality based on historical attention scores, selecting tokens within a limited budget. However, these methods permanently discard parts of the KV Cache, causing information loss from long contexts. To address this, InfLLM (Xiao et al., 2024a) introduces Block Memory Units for KV Cache management, retrieving information from long contexts and offloading less-used blocks to CPU. Similarly, QUEST (Tang et al., 2024) proposes query-aware sparsity at page granularity, while MInference (Jiang et al., 2024) optimizes long-context inference using three sparse patterns. Apart from considering all attention heads, some other works (Ribar et al., 2024; Lee et al., 2024) attempt to focus on only a subset of attention heads. While existing methods have shown progress, opportunities for further improvement remain in achieving optimal accuracy and computational efficiency for real-world deployment.

3 Preliminaries

As discussed in the *Sec.* 1, the high attention sparsity in LLMs suggests sparse attention as a promising solution for long-context inference challenges, which can keep the number of tokens participating in attention computations at a constant scale. Given that predefined sparse patterns are detrimental to performance, we aim to dynamically select crucial tokens at each step during the inference process. Accordingly, based on the overview of LLM inference presented in Appendix D, we formalize the Selective Sparse Attention Problem as follows.

Definition 1 *(Selective Sparse Attention Problem, informal). For current input of length C ($C = 1$ in the decode stage) and KV Cache of length N, assuming there are H attention heads with size of d_h, let $\mathbf{O}$ be the output of the SDPA:*

$$\mathbf{O} = \left[\sigma\left(\frac{\mathbf{Q}^h \cdot [\mathbf{K}^h_{\text{cache}}, \mathbf{K}^h_{\text{current}}]^\top}{\sqrt{d}}\right) \cdot [\mathbf{V}^h_{\text{cache}}, \mathbf{V}^h_{\text{current}}]\right]_{h=1}^{H}, \tag{1}$$

where σ denotes softmax, $\mathbf{Q}^h, \mathbf{K}^h_{\text{current}}, \mathbf{V}^h_{\text{current}} \in \mathbb{R}^{C\times d_h}$ are Query, Key, Value matrices of current input for head h and $\mathbf{K}^h_{\text{cache}}, \mathbf{V}^h_{\text{cache}} \in \mathbb{R}^{N\times d_h}$ represent the KV Cache. Let $\hat{\mathbf{O}}$ be the output of the Selective Sparse Attention:

$$\hat{\mathbf{O}} = \left[\sigma\left(\frac{\mathbf{Q}^h \cdot [\mathbf{K}^h_{\text{select}}, \mathbf{K}^h_{\text{current}}]^\top}{\sqrt{d}}\right) \cdot [\mathbf{V}^h_{\text{select}}, \mathbf{V}^h_{\text{current}}]\right]_{h=1}^{H}, \tag{2}$$

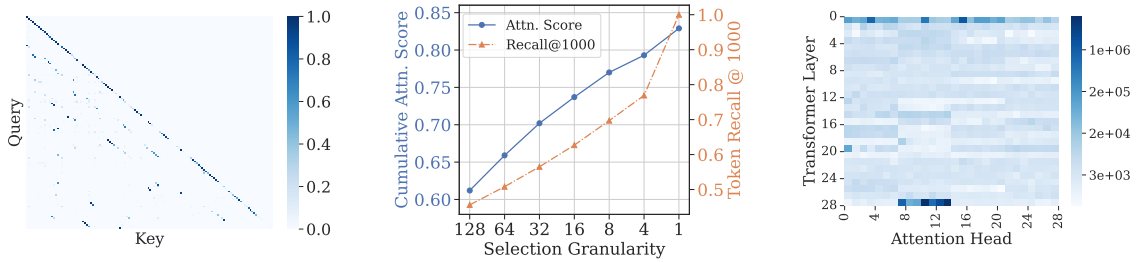
where $\mathbf{K}^h_{\text{select}}, \mathbf{V}^h_{\text{select}} \in \mathbb{R}^{k\times d_h}$ are k selected KV Cache ($k \ll N$). The selection of $\mathbf{K}_{\text{select}}, \mathbf{V}_{\text{select}}$ is performed by selection function $\mathcal{S}$:

$$\begin{aligned} &\mathcal{S}(\mathbf{Q}, \mathbf{K}_{\text{cache}}) = \mathcal{I}, \text{ where } \mathcal{I} \in \mathcal{P}(\{1, \cdots, N\}), \\ &\mathbf{K}_{\text{select}} = [(\mathbf{K}_{\text{cache}})_i]_{i\in\mathcal{I}}, \ \mathbf{V}_{\text{select}} = [(\mathbf{V}_{\text{cache}})_i]_{i\in\mathcal{I}}, \end{aligned} \tag{3}$$

where $\mathcal{I}$ is the set of selected indices. The objective is to find an appropriate selection function $\mathcal{S}$ that minimizes the difference between the outputs of the SDPA and the selective sparse attention:

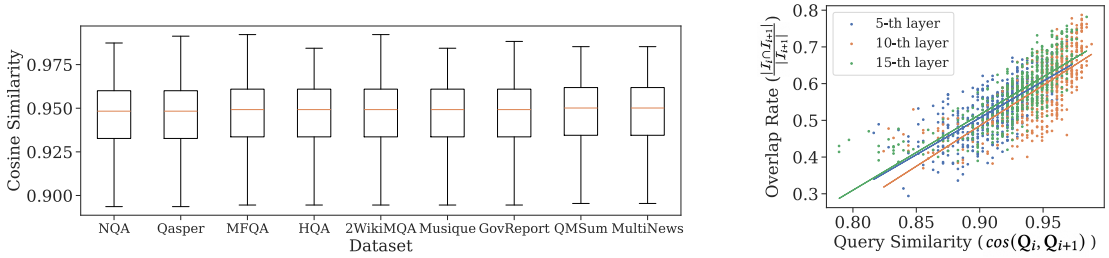
$$\min_{S} \left\|\mathbf{O} - \hat{\mathbf{O}}\right\|_2^2. \tag{4}$$

Existing works on long-context inference can be categorized under the Selective Sparse Attention Problem, with variations in the design of the selection function $\mathcal{S}$. Big-Bird and StreamLLM have developed input-independent selection functions $\mathcal{S}()$, while H_2O, TOVA and SnapKV propose Query-independent functions $\mathcal{S}(\mathbf{K}_{\text{cache}})$ for improved performance. Current state-of-the-art methods InfLLM, QUEST and MInference utilize Query-aware selection functions $\mathcal{S}(\mathbf{Q}, \mathbf{K}_{\text{cache}})$. However, these approaches typically select at a block-level, which limits their effectiveness.



(a) Attention is sparse in token-level. (b) Block-level selection is sub-optimal. (c) Attention logits is head-distinctive.

Figure 2: Motivations for token-level selection. (a) Visualization of attention scores sparsity. (b) Attention scores and critical token recalled by 1K token budget. (c) The L_1 norm of attention logits in each attention head.



(a) Consecutive queries show consistent similarity patterns across datasets. (b) Selection overlaps with similar queries.

Figure 3: Observations on similarity of consecutive queries. (a) Cosine similarity distribution between consecutive queries. (b) The token selection overlap rate ($\frac{|\mathcal{I}_i \cap \mathcal{I}_{i+1}|}{|\mathcal{I}_{i+1}|}$) with respect to consecutive Query similarity.

4 Motivations and Observations

Attention is Sparse, Non-contiguous and Head-Distinctive. Previous works on long-context inference have demonstrated the sparsity of attention scores in LLMs, particularly when processing long texts. Recent approaches (*e.g.*, InfLLM, QUEST and MInference) partition the KV Cache into non-overlapping blocks, estimating block criticality for sparse attention calculations. These methods assume that critical tokens tend to be contiguous. However, our further observations reveal that this assumption does not always hold true in practice. As illustrated in *Fig.* 2a, attention scores are sparsely distributed at the token-level. This non-contiguity leads to significant omissions in block-level token selection. *Fig.* 2b demonstrates that finer selection granularity improves recall of critical tokens, motivating us to perform token-level selection. For token-level selection, an intuitive approach would be to directly select the top-k tokens with largest attention logits. However, *Fig.* 2c reveals considerable disparity in the L_1 norm of attention logits across attention heads. As a result, the selection result tends to be dominated by a few heads with disproportionately large attention logits, driving us to design a more robust selection function that maintains the independence of heads.

Consecutive Queries are Similar. As sparsity of attention is dynamic (Jiang et al., 2024), token selection should be performed for every Query, which inevitably increases the computational overhead of selective sparse attention. Fortunately, we observe that consecutive Queries exhibit high similarity, as shown in *Fig.* 3a. Intuitively, when two consecutive Queries are highly similar, their dot products with the Keys will also be similar, leading to substantial overlap in the token selection results. Due to space constraints, we provide an informal lemma about this below. The formal version and corresponding proof can be found in the Appendix C.

Lemma 1 *(Informal). Consider Queries* $\mathbf{Q}_1, \mathbf{Q}_2 \in \mathbb{R}^{1\times d}$ *that are consecutive and a Key set* $\{\mathbf{K}_i\}_{i=1}^{N}$. *Let* $\mathcal{I}_1$, *and* $\mathcal{I}_2$ *be the sets of indices of the top-*k *Keys selected by dot product for* $\mathbf{Q}_1$, *and* $\mathbf{Q}_2$ *respectively. If* $cos(\mathbf{Q}_1, \mathbf{Q}_2) > \epsilon$, *where* ϵ *is a threshold, then* $\mathcal{I}_1 = \mathcal{I}_2$.

Fig. 3b illustrates this lemma experimentally. It shows that the overlap rate of token selection tends to increase with Query similarity. This key insight motivates us to reuse selection results for similar queries, improving computational efficiency. Moreover, the similarity distribution of consecutive

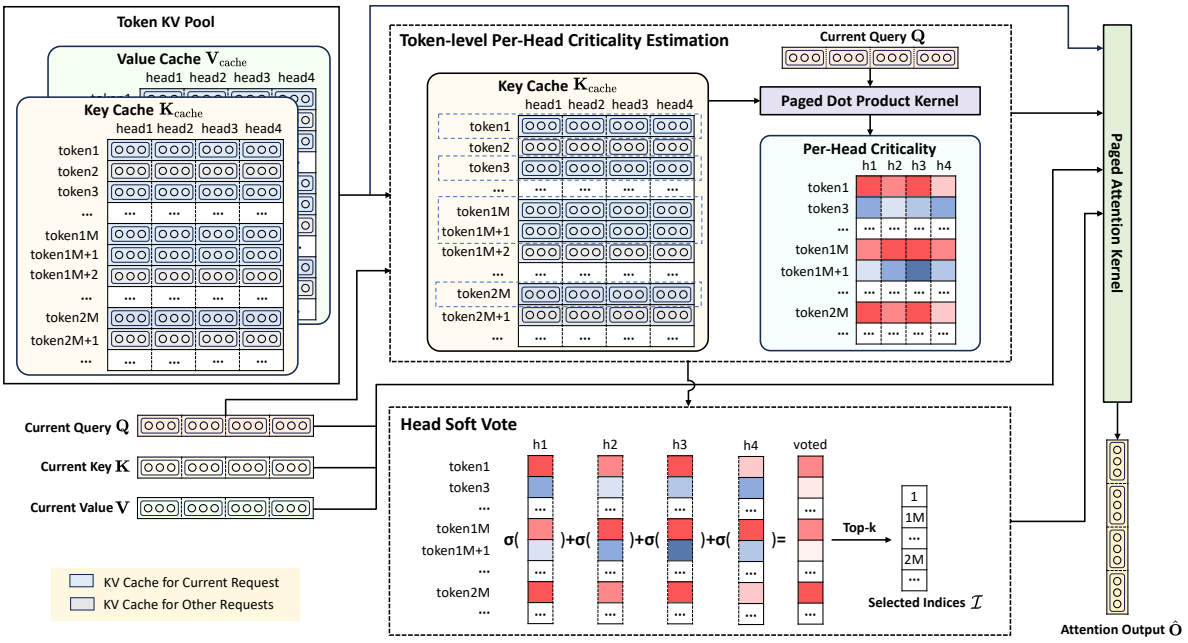


Figure 4: Execution flow of *TokenSelect*: 1) calculate per-head criticality via Paged Dot Product Kernel; 2) perform head soft vote to obtain selected indices; 3) execute selective sparse attention via Paged Attention Kernel.

Queries remains consistent across different tasks, as demonstrated in *Fig.* 3a, allowing us to apply a global similarity threshold across all scenarios.

5 Designs of *TokenSelect*

In this section, we will introduce the design details of *TokenSelect*, primarily encompassing the Selection Function, the Selection Cache, and efficient implementation of *TokenSelect*. The overall workflow of *TokenSelect* is illustrated in *Fig.* 4.

5.1 Selection Function

The simplest selection function is to determine the criticality of the tokens via the dot product of $\mathbf{Q}$ and $\mathbf{K}_{\text{cache}}$, then select the top-$k$ as $\mathbf{K}_{\text{select}}, \mathbf{V}_{\text{select}}$. The selected indices $\mathcal{I}$ are calculated as:

$$\mathcal{I}_{\text{topk}} = \text{TopK}\left(\mathbf{Q} \cdot {\mathbf{K}^h_{\text{cache}}}^\top\right). \tag{5}$$

However, as discussed in *Sec.* 4, this approach is prone to inaccuracies due to disparities in norm of attention logits between heads. To maintain independence between heads, a better approach is to have each head select the top-k most critical tokens, and then determine the final selection through voting among the heads, where $\mathbb{I}$ is indicator function:

$$\mathcal{I}_{\text{head-vote}} = \text{TopK}\left(\sum_{h=1}^{H} \mathbb{I}\left(i \in \text{TopK}\left(\mathbf{Q}^h \cdot {\mathbf{K}^h_{\text{cache}}}^\top\right)\right)\right) \tag{6}$$

Unfortunately, despite better performance, this method relies on `scatter_add` and multiple `topk` operations, resulting in low efficiency on GPUs. Additionally, the 0/1 voting ignores the relative importance of tokens for each head. Therefore, we propose a head soft vote approach that offers better performance and efficiency. Specifically, we first calculate the per-head criticality, then normalize through softmax, and sum the results for all heads:

$$\mathcal{I}_{\text{head-soft-vote}} = \text{TopK}\left(\sum_{h=1}^{H} \sigma\left(\mathbf{Q}^h \cdot {\mathbf{K}^h_{\text{cache}}}^\top\right)\right). \tag{7}$$

5.2 Optimizing Selection Frequency

Although the aforementioned selection function can reduce the complexity of attention from $O(N^2)$ to $O(k^2), k \ll N$, while maintaining performance, the execution time of the selection function itself still affects the latency of inference. To further accelerate long-context inference, based on our observations of the similarity of consecutive Queries, we design optimization strategies for both the prefill stage and the decode stage to reduce the selection frequency while ensuring its effectiveness.

In the prefill stage, $\mathbf{Q}_{\text{prefill}} \in \mathbb{R}^{n_{\text{in}} \times d}$ is inputed. In long-context scenarios, the number of tokens in the user's input sequence n_{in} may reach up to 1M, making it impractical to perform selection for each Query token. Considering the similarity of consecutive Queries, we use chunk-wise token selection, inputting $\frac{1}{c}\sum_{i=1}^{c}(\mathbf{Q}_C)_i$ into the selection function, where $\mathbf{Q}_C \in \mathbb{R}^{c \times d}$ is the Query chunk and c is the chunk size. This method helps maintain the compute-intensive nature of the prefill stage, preventing it from becoming memory bound.

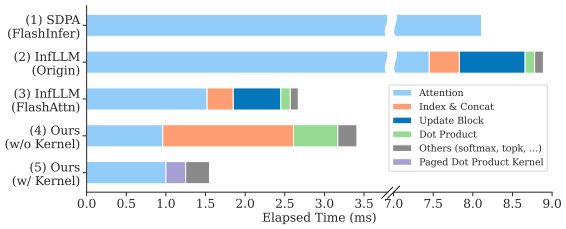


Figure 5: Time breakdown for single chunk prefill step under different attention implementations (chunk size: 512, KV Cache length: 128K, attended tokens: 4K).

In the decode stage, due to the auto-regressive characteristic of LLMs, we need to frequently perform selection for $\mathbf{Q}_{\text{decode}}$, and this process cannot be executed chunk-wise like in the prefill stage. To reduce the frequency of token selection in the decode stage, we propose the Selection Cache. Consecutive similar Queries will hit the cache, thereby directly loading the cached selection results for the previous Query. The Selection Cache allows us to reduce decode latency while maintaining the performance. The formal formulation of the Selection Cache is detailed in Algorithm 1.

5.3 Efficient Implementation

To ready *TokenSelect* for real-world use, efficient implementation is crucial. We first analyze the time breakdown of representative block-level selective sparse attention method, InfLLM (Xiao et al., 2024a). From (1)(2)(3) in *Fig.* 5, we can observe that, despite lowering theoretical complexity, actual runtime depends heavily on implementation. The incompatibility with efficient attention implementations such as Flash Attention has resulted in methods requiring historical attention scores (*e.g.*, H_2O, TOVA, SnapKV, InfLLM) impractical in real-world serving. Analysis of InfLLM's Flash Attention–compatible version shows that, although block-level criticality estimation aims to cut selection overhead, the dot product isn't the main bottleneck. Instead, indexing and coalescing selected KV Cache tokens in GPU memory (HBM)—during block updates and KV Cache concatenation—incurs heavy I/O, aggravating LLM inference's memory-bound limits. Based on this, we propose that Paged Attention is a more suitable implementation for selective sparse attention. Using paged KV Cache management (with page size=1 for *TokenSelect*), we can reduce the I/O volume for selection results from the scale of all selected KV Caches $O(2kd)$ to the scale of their indices $O(k)$. However, (4) in *Fig.* 5 reveals another bottleneck under paged KV Cache management. Since logically contiguous KV Cache is not entirely contiguous in HBM, it also needs to be made contiguous before performing selection operations. To address this issue, we design a Paged Dot Product Kernel using Triton, which significantly improves the overall efficiency of *TokenSelect*. The formal description of this kernel is detailed in Algorithm 2.

6 Experiments

In this section, we introduce the experimental setup and evaluate the performance and efficiency of our *TokenSelect* on long-context inference benchmarks.

6.1 Experimental Settings

Datasets. To evaluate *TokenSelect*'s performance on long-context inference, we use three representative datasets: InfiniteBench (Zhang et al., 2024a), RULER (Hsieh et al., 2024), and LongBench (Bai et al., 2024). Detailed descriptions and the evaluation metrics used are provided in Appendix G.

Baselines. To conduct a comprehensive evaluation of *TokenSelect*'s performance, we carry out benchmarks on three mainstream open-source LLMs-`Qwen2-7B-Instruct` (Yang et al., 2024a), `Llama-3-8B-Instruct` (Dubey et al., 2024), and `Yi-1.5-6B-Chat` (AI et al., 2024)-comparing against the following state-of-the-art long-context inference methods: NTK-scaled RoPE, SelfExtend, StreamingLLM, InfLLM, *SnapKV*, *InfiniGen*, *QUEST*, *RetrievalAttention* and *MInference*. Detailed descriptions of these methods are provided in Appendix F. It is worth noting that the methods indicated in *italics* lack length-extrapolation capability; thus, we evaluate them using an alternative approach, applying them to `Llama-3-8B-Instruct-262k` (long-text post-trained `Llama-3-8B-Instruct`).

Implementation details. In all experiments in this paper, we employ greedy decoding to ensure the reliability of the results. For our *TokenSelect*, we implement it on SGLang (Zheng et al., 2024), which is a fast serving framework based on Flasher-infer (flashinfer ai). We implement our method using PyTorch (Paszke et al., 2019) and Triton (Tillet et al., 2019). We follow the baseline approach, including 128 initial tokens and n_{local} most recent tokens in the attention computation in addition to the k selected tokens. For NTK and SelfExtend, we extend the model's context length to 128K. For

Methods	En.Sum	En.QA	En.MC	En.Dia	Code.D	Math.F	R.PK	R.Num	R.KV	Avg.
Qwen2-7B	23.80	14.92	54.59	8.50	28.17	19.71	28.81	28.64	19.00	25.13
NTK	18.73	15.34	41.28	**7.50**	24.87	**27.71**	99.15	97.46	59.80	43.54
SelfExtend	3.76	4.44	20.09	5.00	8.12	2.29	0.00	0.00	0.00	4.86
StreamingLLM	19.60	13.61	48.03	3.50	27.92	19.43	5.08	5.08	2.40	16.07
InfLLM	19.65	15.71	46.29	**7.50**	27.41	24.00	70.34	72.20	5.40	32.06
TokenSelect	**22.62**	**18.86**	**54.31**	**7.50**	**30.20**	21.71	**100.00**	**100.00**	**86.60**	**49.08**
Llama-3-8B	24.70	15.50	44.10	7.50	27.92	21.70	8.50	7.80	6.20	18.21
NTK	6.40	0.40	0.00	0.00	0.50	2.60	0.00	0.00	0.00	1.10
SelfExtend	14.70	8.60	19.70	0.00	0.00	22.60	**100.00**	**100.00**	0.20	29.53
StreamingLLM	20.40	14.30	40.60	5.00	**28.43**	21.40	8.50	8.30	0.40	16.37
InfLLM	24.30	19.50	43.70	**10.50**	27.41	23.70	**100.00**	99.00	5.00	39.23
TokenSelect	**26.99**	**21.32**	**45.85**	8.00	27.41	**28.29**	**100.00**	97.29	**48.40**	**43.90**
Yi-1.5-6B	18.78	10.48	39.74	5.00	29.95	16.00	5.08	5.08	0.00	14.45
NTK	4.66	0.58	0.87	0.00	0.00	1.43	0.00	0.00	0.00	0.83
SelfExtend	5.62	1.07	1.31	0.00	0.00	1.14	0.00	0.00	0.00	1.01
StreamingLLM	15.35	9.26	35.81	5.00	27.41	14.29	5.08	4.92	0.00	13.01
InfLLM	16.98	8.93	34.06	3.00	27.41	16.86	**100.00**	96.61	0.00	33.76
TokenSelect	**21.13**	**12.32**	**40.61**	**5.50**	**30.71**	**20.86**	**100.00**	**99.83**	0.00	**36.77**

Table 1: Comparison of different methods with different origin models on InfiniteBench.

Methods	4K	8K	16K	32K	64K	128K	Avg.
Qwen2-7B	90.74	84.03	80.87	79.44	74.37	64.13	78.93
StreamingLLM	94.41	54.59	33.54	22.40	15.38	10.88	38.53
InfLLM (2K+512)	52.85	36.09	29.36	23.52	18.81	18.29	29.82
InfLLM (4K+4K)	55.22	52.10	40.53	29.77	21.56	18.64	36.30
Ours (2K+512)	94.11	81.81	68.68	60.62	51.81	42.75	66.63
Ours (4K+4K)	**94.42**	**90.22**	**82.06**	**70.40**	**59.66**	**54.28**	**75.17**
Llama-3-8B	93.79	90.23	0.09	0.00	0.00	0.00	30.69
StreamingLLM	93.68	54.48	33.77	20.35	14.88	11.47	38.11
InfLLM (2K+512)	79.79	52.43	40.12	33.60	25.68	23.39	42.50
InfLLM (4K+4K)	93.79	86.11	64.33	45.39	33.13	27.81	58.43
Ours (2K+512)	93.73	82.92	**71.92**	**65.38**	**59.35**	33.39	**67.78**
Ours (4K+4K)	**93.88**	**90.29**	70.13	57.72	48.36	**39.38**	66.63
Yi-1.5-6B	73.12	9.09	0.37	0.01	0.00	0.01	13.77
StreamingLLM	72.10	33.03	21.69	15.39	12.58	12.61	27.90
InfLLM (2K+512)	59.66	36.77	27.41	24.49	21.49	21.17	31.83
InfLLM (4K+4K)	74.81	52.57	27.65	22.83	20.19	19.48	36.26
Ours (2K+512)	**75.93**	**59.55**	**49.69**	**42.36**	**34.68**	**31.36**	**48.93**

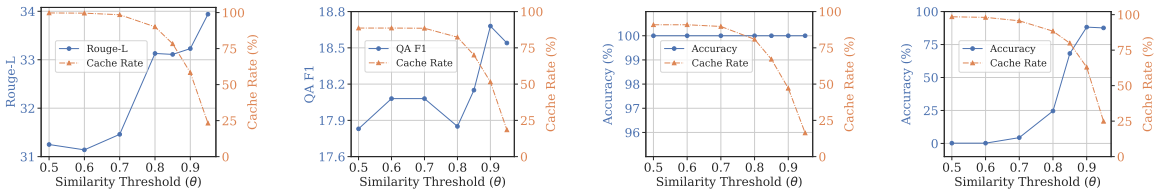
Table 2: Performance comparison on RULER.

StreamLLM, we set $n_{\text{local}} = 4\text{K}$. For InfLLM, we set $k = 4\text{K}, n_{\text{local}} = 4\text{K}$. For our *TokenSelect*, we set $k = 2\text{K}, n_{\text{local}} = 512$ to demonstrate our token-level KV Cache selection allows us to achieve better performance with a smaller token budget. Due to the need to demonstrate the method under different n_{local} and k, we denote the specific token budgets in the form of $k + n_{\text{local}}$ if they differ from the aforementioned settings. For InfiniteBench and LongBench, we set the threshold θ of the Selection Cache to 0.9. We use NVIDIA A100 to conduct all experiments. When inferencing sequences over 1M tokens, we additionally employee tensor parallelism, which is transparent to our *TokenSelect*.

6.2 Performance Comparisons

InfiniteBench. As shown in Table 1, our *TokenSelect* achieves significantly superior overall performance on InfiniteBench compared to all baseline methods, even though *TokenSelect* uses the smallest token budget (<3K). The fact that it significantly outperforms the original models demonstrates *TokenSelect*'s strong length extrapolation capability. We analyze that this is due to our adoption of a fine-grained KV Cache selection strategy, while considering the equal contribution of each head to selection, which ensures that we can select most critical tokens. Observing the performance of other methods, we find that RoPE interpolation methods (NTK, SelfExtend) generally perform poorly unless used on specially trained models such as `Qwen2-7B-Instruct`. The sparse attention method StreamingLLM, based on fixed sparse patterns, can guarantee some of the model's capabilities, but due to discarding a large amount of long-context information, it performs poorly on retrieval-related tasks (R.PK, R.Num, R.KV). The block-level selection method InfLLM can retain more long-context information compared to StreamingLLM. However, due to its sub-optimal block-level selection, it results in lower performance on most tasks compared to *TokenSelect*, even though we set a larger token budget for InfLLM. It is worth noting that `Yi-1.5-6B` does not perform normally on the R.KV task, as it is unable to correctly recite strings like the UUID.

RULER. To further demonstrate the capability of *TokenSelect*, we conduct evaluation on the more challenging long-context benchmark RULER. Considering the increased difficulty of RULER and its substantial computational requirements, we include only comparable baseline methods. As shown in Table 2, our *TokenSelect* maintains significantly superior overall performance compared to other long-context inference methods. For all models, *TokenSelect* achieves length extrapolation while preserving the model's original capabilities, benefiting from our efficient utilization of the model's limited context length. Notably, due to the constraints of



(a) LongBench-GovReport. (b) InfiniteBench-En.QA. (c) InfiniteBench-R.PK. (d) InfiniteBench-R.KV.

Figure 6: Performance and Cache Rate with different threshold θ of the Selection Cache on `Qwen2-7B-Instruct`.

Methods	En.QA	En.MC	Code.D	R.PK	R.Num	R.KV
Llama-3-8B-Instruct-262k						
SDPA (128K)	9.10	68.00	19.00	100.00	100.00	17.50
SDPA (262K)	12.40	67.30	22.10	100.00	100.00	14.40
StreamingLLM (2K+512)	6.00	66.00	18.50	5.00	5.00	1.00
SnapKV (2K+512)	11.80	67.00	18.00	100.00	100.00	0.50
InfLLM (2K+512)	7.00	37.00	20.50	100.00	100.00	0.50
InfiniGen (2K+512)	7.30	57.50	17.50	100.00	99.50	0.00
QUEST (2K+512)	8.20	67.00	18.00	100.00	100.00	0.00
RetrievalAttn. (2K+512)	7.50	67.00	19.00	100.00	100.00	14.00
MInference w/ static	8.60	43.20	20.60	92.40	96.30	0.20
MInference	12.90	65.90	22.30	100.00	100.00	12.80
Ours (2k+512)	9.70	**68.00**	19.00	**100.00**	**100.00**	20.60
Llama-3-8B-Instruct						
Ours (2k+512)	**21.32**	45.85	**27.41**	**100.00**	97.29	**48.40**

Table 3: Performance comparison with methods based-on post-trained models. Baseline performance is referenced from Jiang et al. (2024) and Liu et al. (2024a).

model's context length, *TokenSelect* experiences performance degradation with larger token budgets (4K+4K) on Llama and Yi. However, its performance with smaller token budgets still significantly surpasses other baseline methods.

Comparing to methods based-on post-trained models. In Table 3, we present a performance comparison of baseline methods that do not support length extrapolation and must be applied to long-text post-trained models. Our results show that, even compared with models undergoing costly long-text post-training and the methods applied to them, the training-free *TokenSelect* exhibits superior performance on most tasks. These findings further demonstrate the effectiveness of *TokenSelect* in long-context inference and length extrapolation.

6.3 Ablation Studies

Selection functions $\mathcal{S}$. To compare the performance of different selection functions $\mathcal{S}$ under low token budgets (*i.e.*, token efficiency), we maintain the 2K+512 configuration. From Table 4, we can observe that our proposed head soft vote mechanism performs significantly better across all tasks. This indicates that using the head soft vote mechanism to balance each head's contribution to token selection results can help us avoid the domination of selection by few heads with large attention logits.

$\mathcal{S}$	En.QA	En.MC	Code.D	R.PK	R.Num	R.KV
$\mathcal{I}_{\text{topk}}$	15.15	45.85	28.43	**100.00**	98.47	16.60
$\mathcal{I}_{\text{head-vote}}$	17.01	45.85	28.68	**100.00**	**100.00**	22.40
$\mathcal{I}_{\text{head-soft-vote}}$	**18.86**	**54.31**	**30.20**	**100.00**	**100.00**	**86.60**

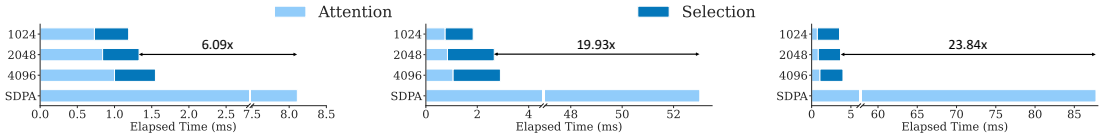
Table 4: Ablation study of the Selection Function $\mathcal{S}$ on InfiniteBench using `Qwen2-7B-Instruct`.

k	En.Sum	En.QA	En.Mc	Math.F	R.Num	R.KV
128	21.23	10.46	41.48	18.00	100.00	13.40
256	22.01	11.66	41.92	19.71	100.00	20.00
512	21.60	13.31	40.17	21.71	100.00	45.60
1K	21.35	15.13	44.10	21.71	100.00	73.00
2K	22.62	18.86	54.31	21.71	100.00	86.60
4K	24.09	21.11	51.53	21.71	100.00	**88.00**
8K	25.32	22.93	58.52	23.71	100.00	85.40
16K	**26.54**	**23.04**	**62.88**	**28.16**	**100.00**	72.00

Table 5: Performance vs. Number of selected tokens k on InfiniteBench using `Qwen2-7B-Instruct`.

Similarity threshold of the Selection Cache θ. *Fig.* 6 shows that the Selection Cache hit rate increases significantly as the similarity threshold θ decreases, converging around $\theta = 0.5$. This suggests potential for further acceleration of *TokenSelect*'s decode stage by reducing θ. Performance sensitivity to θ varies across tasks. While most tasks exhibit slight performance degradation with decreasing θ, and R.PK in InfiniteBench shows no degradation, more challenging retrieval tasks like R.KV demonstrate significant performance deterioration. This indicates higher dynamicity requirements for token selection in these tasks. Owing to the limited generation lengths in current long-context inference benchmarks, we cannot yet precisely quantify the end-to-end speedup provided by the Selection Cache. Nonetheless, for a 7B-parameter model operating on 128K-token sequences, each cache hit reduces per-step latency by approximately 0.5 ms. For more detailed performance comparisons under different θ, see Table 9 of Appendix I.

Number of selected tokens k. As shown in Table 5, we fix n_{local} to a small value (512) to compare the performance when selecting different numbers of tokens. First, we observe that even selecting a very small number of tokens (*e.g.*, 128, 256), our



(a) KV Cache length: 128K. (b) KV Cache length: 512K. (c) KV Cache length: 1M.

Figure 7: Computation time v.s. KV Cache lengths for single chunk prefill step using `Qwen2-7B-Instruct`. The vertical axis represents the number of attended tokens. SDPA denotes full attention by Flashinfer (chunk size: 512).

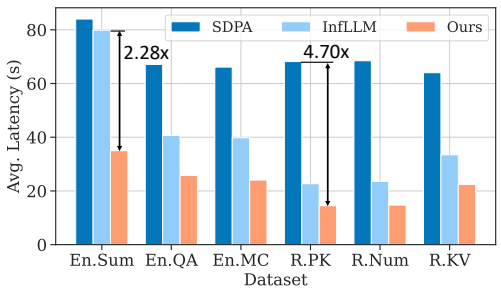


Figure 8: End to end latency per sample with different methods on InfiniteBench using `Qwen2-7B-Instruct`.

TokenSelect still demonstrates very comparable performance. Then, as k increases, the effectiveness of *TokenSelect* further improves, indicating that more moderately critical tokens also contribute to the retention of long-context information. Finally, we find that when k is set to larger values (*e.g.*, 16K), our *TokenSelect* shows significant improvements in most tasks, further advancing the performance landscape of long-context inference methods.

6.4 Efficiency Comparisons

Efficiency of selective sparse attention. *Fig.* 7 demonstrates the significant acceleration of attention computation achieved by *TokenSelect* during long-context inference. With a KV Cache length of 1M, *TokenSelect* can provide up to 23.84× speedup compared to FlashInfer, which is the inference kernel library we based on. This substantial improvement is attributed to our efficient kernel design.

End-to-end efficiency. *Fig.* 8 compares the end-to-end latency of *TokenSelect*, InfLLM, and SDPA across various tasks. *TokenSelect* significantly accelerates long-context inference in real-world scenarios, achieving a maximum speedup of 4.70× over SDPA and 2.28× over the state-of-the-art long-context inference method while also delivering superior overall performance.

7 Conclusion

In this paper, we introduces *TokenSelect*, a training-free approach for efficient long-context inference and length extrapolation. *TokenSelect* addresses the two major challenges faced by LLMs in processing long texts: the context length limitation from pre-training and the computational complexity of attention. This is achieved through a novel token-level selective sparse attention mechanism. Experimental results demonstrate that *TokenSelect* can achieve up to 23.84× speedup in attention computation and up to 2.28× acceleration in end-to-end inference latency, while exhibiting superior performance across multiple long-context benchmarks.

8 Limitations

Our approach has inherent limitations that present opportunities for future work. A primary limitation of our method is that its training-free design—a significant advantage—acts as a double-edged sword, as its absolute performance is inherently tied to the quality of the underlying LLMs. Although our experiments demonstrate robustness of *TokenSelect* across various LLMs, some inherent shortcomings—such as the misrecognition of UUID strings by `Yi-1.5-6B-Chat`—indicate that certain issues may still require training to resolve. Moreover, while our method currently achieves state-of-the-art performance in long-context inference, recent long-text post-training techniques in the LLM community have shown impressive performance; notably, our *TokenSelect* is orthogonal to these approaches and can be employed during inference to trade a slight performance drop for significant efficiency gains. Finally, although our method achieves state-of-the-art efficiency improvements in long-context inference, the task remains inherently resource-intensive. For instance, even with a 8B-parameter model, complex benchmarks (e.g., RULER) can require approximately 8×A100 GPUs for nearly one day of runtime, and the computational cost is expected to increase substantially for larger models. We hope that our work, together with the community's advances in model design, algorithm development, and infrastructure optimization, will help pave the way for further mitigating these computational challenges.

Acknowledgment

This work was supported in part by the National Key R&D Program of China (Grant No.2023YFF0725001), in part by the National Natural Science Foundation of China (Grant No.92370204), in part by the Guangdong Basic and Applied Basic Research Foundation (Grant No.2023B1515120057), in part by the Education Bureau of Guangzhou Municipality.

References

01. AI, :, Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, Kaidong Yu, Peng Liu, Qiang Liu, Shawn Yue, Senbin Yang, Shiming Yang, Tao Yu, and 13 others. 2024. Yi: Open foundation models by 01.ai. Preprint, arXiv:2403.04652.

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. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

bloc97. 2023. Ntk-aware scaled rope allows llama models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation. Website. https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. 2022. Recurrent memory transformer. Advances in Neural Information Processing Systems, 35:11079–11091.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. Advances in Neural Information Processing Systems, 37:37665–37691.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. Preprint, arXiv:2306.15595.

Zihang Dai*, Zhilin Yang*, Yiming Yang, William W. Cohen, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Language modeling with longer-term dependency.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In International Conference on Learning Representations (ICLR).

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, and 1 others. 2024. The llama 3 herd of models. Preprint, arXiv:2407.21783.

emozilla. 2023. Dynamically scaled rope further increases performance of long context llama with zero fine-tuning. Website. https://www.reddit.com/r/LocalLLaMA/comments/14mrgpr/dynamically_scaled_rope_further_increases/.

flashinfer ai. GitHub - flashinfer-ai/flashinfer: FlashInfer: Kernel Library for LLM Serving — github.com. https://github.com/flashinfer-ai/flashinfer. [Accessed 12-10-2024].

Team GLM, :, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, and 1 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. Preprint, arXiv:2406.12793.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. LM-infinite: Zero-shot extreme length generalization for large language models. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 3991–4008, Mexico City, Mexico. Association for Computational Linguistics.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the real context size of your long-context language models? In First Conference on Language Modeling.

Yunpeng Huang, Jingwei Xu, Junyu Lai, Zixu Jiang, Taolue Chen, Zenan Li, Yuan Yao, Xiaoxing Ma, Lijuan Yang, Hao Chen, Shupeng Li, and Penghao Zhao. 2024. Advancing transformer architecture in long-context large language models: A comprehensive survey. Preprint, arXiv:2311.12351.

Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Reza Yazdani Aminadabi, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2024. System optimizations for enabling training of extreme long sequence transformer models. In Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC '24, page 121–130, New York, NY, USA. Association for Computing Machinery.

Arthur Jacot, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: Convergence and generalization in neural networks. Advances in neural information processing systems, 31.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix,

and William El Sayed. 2023. Mistral 7b. Preprint, arXiv:2310.06825.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, and 1 others. 2024. Minference 1.0: Accelerating prefilling for long-context llms via dynamic sparse attention. Advances in Neural Information Processing Systems, 37:52481–52515.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In Proceedings of the 29th symposium on operating systems principles, pages 611–626.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. Infinigen: efficient generative inference of large language models with dynamic kv cache management. In Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation, OSDI'24, USA. USENIX Association.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. Advances in Neural Information Processing Systems, 37:22947–22970.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, and 1 others. 2024a. Retrievalattention: Accelerating long-context llm inference via vector retrieval. arXiv preprint arXiv:2409.10516.

Hao Liu, Matei Zaharia, and Pieter Abbeel. 2024b. Ringattention with blockwise transformers for near-infinite context. In The Twelfth International Conference on Learning Representations.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. 2024. Leave no context behind: Efficient infinite context transformers with infini-attention. Preprint, arXiv:2404.07143.

OpenAI. Introducing OpenAI o1. https://openai.com/o1/. [Accessed 06-10-2024].

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state RNNs. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 18724–18741, Miami, Florida, USA. Association for Computational Linguistics.

Arka Pal, Deep Karkhanis, Manley Roberts, Samuel Dooley, Arvind Sundararajan, and Siddartha Naidu. 2023. Giraffe: Adventures in expanding context lengths in llms. Preprint, arXiv:2308.10882.

Zhuoshi Pan, Yu Li, Honglin Lin, Qizhi Pei, Zinan Tang, Wei Wu, Chenlin Ming, H. Vicky Zhao, Conghui He, and Lijun Wu. 2025. LEMMA: Learning from errors for MatheMatical advancement in LLMs. In Findings of the Association for Computational Linguistics: ACL 2025, pages 11615–11639, Vienna, Austria. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. YaRN: Efficient context window extension of large language models. In The Twelfth International Conference on Learning Representations.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In International Conference on Learning Representations.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. 2024. Sparq attention: Bandwidth-efficient LLM inference. In Forty-first International Conference on Machine Learning.

Nikhil Sharma, Q. Vera Liao, and Ziang Xiao. 2024. Generative echo chamber? effect of llm-powered search systems on diverse information seeking. In Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, CHI '24, New York, NY, USA. Association for Computing Machinery.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-lm: Training multi-billion parameter language models using model parallelism. Preprint, arXiv:1909.08053.

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

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2025. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. In Forty-second International Conference on Machine Learning.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. QUEST: Query-aware sparsity for efficient long-context LLM inference. In Forty-first International Conference on Machine Learning.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. Preprint, arXiv:2403.05530.

Junfeng Tian, Da Zheng, Yang Chen, Rui Wang, Colin Zhang, and Debing Zhang. 2025. Untie the knots: An efficient data augmentation strategy for long-context pre-training in language models. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1223–1242, Vienna, Austria. Association for Computational Linguistics.

Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL 2019, page 10–19, New York, NY, USA. Association for Computing Machinery.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. Preprint, arXiv:2307.09288.

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage bert: A globally normalized bert model for open-domain question answering. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 5878–5882.

Wei Wu, Qiuyi Li, Mingyang Li, Kun Fu, Fuli Feng, Jieping Ye, Hui Xiong, and Zheng Wang. 2025a. Generator: A long-context generative genomic foundation model. Preprint, arXiv:2502.07272.

Wei Wu, Chao Wang, Liyi Chen, Mingze Yin, Yiheng Zhu, Kun Fu, Jieping Ye, Hui Xiong, and Zheng Wang. 2025b. Structure-enhanced protein instruction tuning: Towards general-purpose protein understanding with llms. In Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2, KDD '25, page 3216–3227, New York, NY, USA. Association for Computing Machinery.

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. Advances in Neural Information Processing Systems, 37:119638–119661.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024b. Efficient streaming language models with attention sinks. In The Twelfth International Conference on Learning Representations.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, and 1 others. 2024a. Qwen2 technical report. Preprint, arXiv:2407.10671.

Shuo Yang, Ying Sheng, Joseph E. Gonzalez, Ion Stoica, and Lianmin Zheng. 2024b. Post-training sparse attention with double sparsity. Preprint, arXiv:2408.07092.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences. In Advances in Neural Information Processing Systems, volume 33, pages 17283–17297. Curran Associates, Inc.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024a. ∞Bench: Extending long context evaluation beyond 100K tokens. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15262–15277, Bangkok, Thailand. Association for Computational Linguistics.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2024b. H2o: Heavy-hitter oracle for efficient generative inference of large language models. Advances in Neural Information Processing Systems, 36.

Liang Zhao, Xiachong Feng, Xiaocheng Feng, Weihong Zhong, Dongliang Xu, Qing Yang, Hongtao Liu, Bing Qin, and Ting Liu. 2023. Length extrapolation of transformers: A survey from the perspective of positional encoding. In Conference on Empirical Methods in Natural Language Processing.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. Sglang: Efficient execution of structured language model programs. Preprint, arXiv:2312.07104.

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 Formal Description of Algorithms

In *Sec.* 5.2, we propose the Selection Cache, which shares selection results among similar Queries to reduce selection frequency without sacrificing performance. Formally, it is defined as follows:

Algorithm 1 Selection Cache Algorithm

Require: $\mathbf{Q} \in \mathbb{R}^{H \times D}$: current query vectors
$k \in \mathbb{N}$: number of tokens to select
$\mathbf{C}_Q \in \mathbb{R}^{H \times D}$: cached query vector
$\mathbf{C}_\mathcal{I} \in \{0, \ldots, N-1\}^k$: cached indices
$\theta \in [0, 1]$: cosine-similarity threshold
$\mathcal{S}$: selection function (Eq. 7)
$f \in \{\text{True}, \text{False}\}$: first-query flag (default True)
Ensure: $\mathcal{I} \in \{0, \ldots, N-1\}^k$: indices of k selected tokens

if f **or** $\cos(\mathbf{Q}, \mathbf{C}_Q) < \theta$ **then**
  $\mathcal{I} \leftarrow \mathcal{S}(\mathbf{Q}, k)$
  $\mathbf{C}_\mathcal{I} \leftarrow \mathcal{I}$
  $\mathbf{C}_Q \leftarrow \mathbf{Q}$
  $\mathbf{f} \leftarrow$ False
else
  $\mathcal{I} \leftarrow C_\mathcal{I}$
end if
return $\mathcal{I}$

In *Sec.* 5.3, we propose the Paged Dot Product Kernel to efficiently perform token-level per-head criticality estimation under the paged KV-cache management by significantly reducing I/O between HBM and SRAM. Formally, it is defined as follows:

Algorithm 2 Paged Dot Product Kernel

Require: $\mathbf{Q} \in \mathbb{R}^{H \times D}$: current query vectors
$\mathbf{K} \in \mathbb{R}^{N_{\text{kv}} \times H_{\text{kv}} \times D}$: key cache pool
$\mathbf{I} \in \{0, \ldots, N_{\text{kv}} - 1\}^T$: indices of relevant tokens
H: number of attention heads
H_{kv}: number of KV heads ($H \bmod H_{\text{kv}} = 0$)
D: head dimension
T: number of relevant tokens ($|\mathbf{I}| = T$)
B: CUDA block size
Ensure: $\mathbf{S} \in \mathbb{R}^{H \times T}$: dot product scores

$N \leftarrow \lceil T/B \rceil$
for all $h = 0, \ldots, H-1$ **in parallel do**
  $q \leftarrow \mathbf{Q}[h, :]$ {to SRAM}
  $h_{\text{kv}} \leftarrow h \bmod H_{\text{kv}}$
  for all $b = 0, \ldots, N-1$ **in parallel do**
    $t_0 \leftarrow b \times B$
    $L \leftarrow \min(B,\ T - t_0)$
    for $j = 0, \ldots, L-1$ **do**
      $idx \leftarrow \mathbf{I}[\,t_0 + j\,]$ {to SRAM}
      $k \leftarrow \mathbf{K}[\,idx,\ h_{\text{kv}}, :\,]$ {to SRAM}
      $s \leftarrow \langle q,\ k \rangle$ {in SRAM}
      $\mathbf{S}[\,h,\ t_0 + j\,] \leftarrow s$ {to HBM}
    end for
  end for
end for
return $\mathbf{S}$

B Scalability of *TokenSelect*

B.1 Scaling Beyond 1 Million Context Length

To further explore *TokenSelect*'s performance in extreme long-context scenarios, we design an extended benchmark with different text lengths following InfiniteBench. As illustrated in the *Fig.* 9, our *TokenSelect* demonstrates the ability to accurately capture critical information with a small token budget in contexts up to 2M tokens, underscoring its potential in more application scenarios.

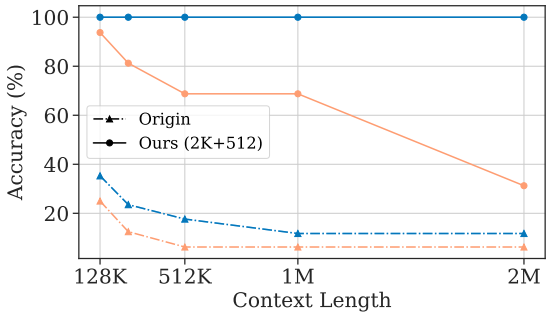


Figure 9: Performance comparison on extended R.PK and R.KV using `Qwen2-7B-Instruct`.

B.2 Scaling to 72 Billion Parameters

To demonstrate the scalability of our approach to larger models, we conducted additional experiments using `Qwen2-72B-Instruct`. The results, presented in Table 6, show that our method outperforms NTK-Aware Scaled RoPE in terms of accuracy and achieves lower latency, indicating the potential of our approach to scale effectively with larger models.

Method	En.Sum		En.QA		R.KV	
	Acc. (%)	Time (s)	Acc. (%)	Time (s)	Acc. (%)	Time (s)
NTK (SPDA)	23.49	199.52	28.77	145.69	50.00	111.98
TokenSelect	**25.07**	**114.24**	**29.91**	**71.98**	**88.12**	**63.27**

Table 6: Performance and latency comparison on `Qwen2-72B-Instruct` with tensor parallelism size: 4.

C Formal Statement and Proof of Lemma

Lemma 1 (Invariant Top-k Key Selection under Cosine Similarity Threshold, Formal).

Assumptions:

1. Let $\mathbf{q}_1, \mathbf{q}_2 \in \mathbb{R}^d$ be two query vectors.
2. Let $\{\mathbf{k}_i\}_{i=1}^N \subset \mathbb{R}^d$ be a finite set of key vectors.
3. Let k be a positive integer such that $1 \leq k \leq N$.
4. Define the cosine similarity between vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ as:
$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}\mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2},$$
where $\|\cdot\|_2$ denotes the Euclidean norm.
5. Define the top-k selection function based on dot product similarity as: $\mathcal{I}(\mathbf{q}) = \arg\max_{S \subseteq \{1,2,\ldots,N\}, |S|=k} \sum_{i \in S} \mathbf{q} \cdot \mathbf{k}_i$. Assume that for any query vectors $\mathbf{q}$, the top-k set $\mathcal{I}(\mathbf{q})$ is uniquely determined.
6. Let $\epsilon \in (0, 1]$ be a predefined threshold.

Lemma Statement: *If the cosine similarity between the two query vectors* $\mathbf{q}_1$ *and* $\mathbf{q}_2$ *satisfies*

$$\cos(\mathbf{q}_1, \mathbf{q}_2) > \epsilon,$$

then the indices of the top-k *keys selected by* $\mathbf{q}_1$ *and* $\mathbf{q}_2$ *are identical, i.e.,*

$$\mathcal{I}(\mathbf{q}_1) = \mathcal{I}(\mathbf{q}_2).$$

Proof: We start with the given condition:

$$\min_{1 \leq i \leq k} \mathbf{q}_1 \mathbf{k}_i - \max_{j > k} \mathbf{q}_1 \mathbf{k}_j > \eta,$$

which we aim to use to demonstrate that:

$$\min_{1 \leq i \leq k} \mathbf{q}_2 \mathbf{k}_i - \max_{j > k} \mathbf{q}_2 \mathbf{k}_j > 0.$$

To facilitate our analysis, we introduce the following notations:

$$\hat{\eta} = \frac{\eta}{\|\mathbf{q}_1\|}, \quad \hat{\mathbf{q}}_1 = \frac{\mathbf{q}_1}{\|\mathbf{q}_1\|}, \quad \hat{\mathbf{q}}_2 = \frac{\mathbf{q}_2}{\|\mathbf{q}_2\|}.$$

With these definitions, the original condition becomes:

$$\min_{1 \leq i \leq k} \hat{\mathbf{q}}_1 \mathbf{k}_i - \max_{j > k} \hat{\mathbf{q}}_1 \mathbf{k}_j > \hat{\eta},$$

and our goal transforms to showing:

$$\min_{1 \leq i \leq k} \hat{\mathbf{q}}_2 \mathbf{k}_i - \max_{j > k} \hat{\mathbf{q}}_2 \mathbf{k}_j > 0.$$

Next, let θ denote the angle between $\mathbf{q}_1$ and $\mathbf{q}_2$, $\cos\theta = \hat{\mathbf{q}}_1 \cdot \hat{\mathbf{q}}_2$. We can further define:

$$\mathbf{p}_1 = \mathbf{q}_2 - \mathbf{q}_1 \cos\theta, \quad \hat{\mathbf{p}}_1 = \frac{\mathbf{p}_1}{\|\mathbf{p}_1\|},$$

then $\sin\theta = \hat{\mathbf{p}}_1 \cdot \hat{\mathbf{q}}_2$, and

$$\hat{\mathbf{q}}_2 = \hat{\mathbf{q}}_1 \cos\theta + \hat{\mathbf{p}}_1 \sin\theta.$$

Then we have:

$$\begin{aligned}
\min_{1 \leq i \leq k} \hat{\mathbf{q}}_2 \mathbf{k}_i &= \min_{1 \leq i \leq k} (\hat{\mathbf{q}}_1 \cos\theta + \hat{\mathbf{p}}_1 \sin\theta) \mathbf{k}_i, \\
&\geq \min_{1 \leq i \leq k} \hat{\mathbf{q}}_1 \mathbf{k}_i \cos\theta + \min_{1 \leq i \leq k} \hat{\mathbf{p}}_1 \mathbf{k}_i \sin\theta, \\
&\geq \hat{\mathbf{q}}_1 \mathbf{k}_k \cos\theta - \|\mathbf{k}\|_{\max} \sin\theta,
\end{aligned}$$

and

$$\begin{aligned}
\max_{j > k} \hat{\mathbf{q}}_2 \mathbf{k}_j &= \max_{j > k} (\hat{\mathbf{q}}_1 \cos\theta + \hat{\mathbf{p}}_1 \sin\theta) \mathbf{k}_j \\
&\leq \max_{j > k} \hat{\mathbf{q}}_1 \mathbf{k}_i \cos\theta + \max_{j > k} \hat{\mathbf{p}}_1 \mathbf{k}_i \sin\theta, \\
&\leq \hat{\mathbf{q}}_1 \mathbf{k}_{p+1} \cos\theta + \|\mathbf{k}\|_{\max} \sin\theta.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\min_{1 \leq i \leq k} \hat{\mathbf{q}}_2 \mathbf{k}_i - \max_{j > k} \hat{\mathbf{q}}_2 \mathbf{k}_j \\
&\quad \geq \hat{\mathbf{q}}_1 \mathbf{k}_p \cos\theta - \|\mathbf{k}\|_{\max} \sin\theta \\
&\qquad\qquad - (\hat{\mathbf{q}}_1 \mathbf{k}_{p+1} \cos\theta \\
&\qquad\qquad + \|\mathbf{k}\|_{\max} \sin\theta) \\
&\quad = (\hat{\mathbf{q}}_1 \mathbf{k}_p \cos\theta - \hat{\mathbf{q}}_1 \mathbf{k}_{p+1} \cos\theta) \\
&\qquad\qquad - 2\|\mathbf{k}\|_{\max} \sin\theta \\
&\quad \geq \hat{\eta} \cos\theta - 2\|\mathbf{k}\|_{\max} \sin\theta. \qquad (8)
\end{aligned}$$

In order to have Eqn. (8) > 0, we require

$$\begin{aligned}
&\hat{\eta} \cos\theta > 2\|\mathbf{k}\|_{\max} \sin\theta, \\
\Rightarrow\ &\frac{\sin\theta}{\cos\theta} < \frac{\hat{\eta}}{2\|\mathbf{k}\|_{\max}}, \\
\Rightarrow\ &\frac{1 - \cos^2\theta}{\cos^2\theta} < \left(\frac{\hat{\eta}}{2\|\mathbf{k}\|_{\max}}\right)^2, \\
\Rightarrow\ &\cos\theta \geq \frac{1}{\sqrt{1 + \left(\frac{\hat{\eta}}{2\|\mathbf{k}\|_{\max}}\right)^2}}.
\end{aligned}$$

This final inequality establishes a sufficient condition for the original statement to hold, thereby completing the proof.

D Overview of LLMs Inference

Nowadays, mainstream LLMs are primarily based on the Decoder-only Transformer architecture. Each transformer layer includes a multi-head attention (MHA) and a feed-forward networks (FFN). The inference process of LLMs can be divided into two stages: the prefill stage and the decode stage.

The prefill stage is the preparatory phase of the inference process. In this stage, the user's input is processed layer by layer through a single forward pass of LLMs, generating KV Cache for each layer. The generation of KV Cache is completed by the MHA module. Assuming $\mathbf{X}_{\text{prefill}} \in \mathbb{R}^{n_{\text{in}} \times d}$ is the input of a transformer layer, where n_{in} is the number of tokens in user's input sequence and d is the hidden size. The MHA computation in the prefill stage is as follows (simplified to single head):

$$[\mathbf{Q}_{\text{prefill}}, \mathbf{K}_{\text{prefill}}, \mathbf{V}_{\text{prefill}}] = \mathbf{X}_{\text{prefill}} \cdot [\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v], \quad (9)$$

$$\mathbf{O}_{\text{prefill}} = \text{softmax}\left(\frac{\mathbf{Q}_{\text{prefill}} \cdot \mathbf{K}_{\text{prefill}}{}^{\top}}{\sqrt{d}}\right) \cdot \mathbf{V}_{\text{prefill}}, \quad (10)$$

where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ are linear projections, $[\cdot]$ represents tensor concatenation operation, and *Eq.*(10) is also known as Scaled Dot Product Attention (SDPA). After these computation, $\mathbf{K}_{\text{prefill}}$ and $\mathbf{V}_{\text{prefill}}$ are stored as the KV Cache for current layer $\mathbf{K}_{\text{cache}}$ and $\mathbf{V}_{\text{cache}}$, and $\mathbf{O}_{\text{prefill}}$ is used for subsequent calculations.

The decode stage is the phase where LLMs actually generate the response. In the decode stage, LLMs load the KV Cache and generate n_{out} output tokens autoregressively through n_{out} forward passes. Assuming $\mathbf{X}_{\text{decode}} \in \mathbb{R}^{1 \times d}$ is the input of a transformer layer in a forward pass, the computation of MHA in the decode stage is as follows (The calculation of $\mathbf{Q}_{\text{prefill}}$ and $\mathbf{O}_{\text{prefill}}$ is consistent with that in the prefill stage):

$$\begin{aligned} \mathbf{K}_{\text{decode}} &= [\mathbf{K}_{\text{cache}},\ \mathbf{X}_{\text{decode}} \cdot \mathbf{W}_k],\ \mathbf{K}_{\text{cache}} \leftarrow \mathbf{K}_{\text{decode}}, \\ \mathbf{V}_{\text{decode}} &= [\mathbf{V}_{\text{cache}},\ \mathbf{X}_{\text{decode}} \cdot \mathbf{W}_v],\ \mathbf{V}_{\text{cache}} \leftarrow \mathbf{V}_{\text{decode}}, \end{aligned} \quad (11)$$

where $\mathbf{K}_{\text{decode}}, \mathbf{V}_{\text{decode}}$ are composed of the KV Cache and the KV corresponding to the current input, which are then used to update the KV Cache of the current layer for use in the next forward pass.

LLMs inference, unlike training, is memory-bound, necessitating frequent GPU I/O operations between HBM and SRAM while underutilizing processing units. This bottleneck is particularly evident in SDPA computation. Optimizing for I/O is crucial for enhancing LLMs inference efficiency, especially in long-context scenarios.

E Comparison with Token Eviction-based Methods (*e.g.*, H_2O)

Token eviction–based methods (Zhang et al., 2024b; Oren et al., 2024), led by H_2O (Zhang et al., 2024b), have pioneered the field of long-context inference, achieving early state-of-the-art performance. Although both our method and H_2O employ token-level criticality estimation, they fall under two entirely different taxonomies. As discussed in *Sec.* 2 and *Sec.* 3, H_2O is a query-independent KV cache selection method, which suffers from three main drawbacks:

1. Lack of dynamism: Its importance scoring relies on attention scores from previous queries and keys. Consequently, KV pairs that are crucial for the current query may have been discarded earlier—a phenomenon also confirmed by QUEST (Tang et al., 2024). *Fig.* 1 and 2 of QUEST provide an intuitive illustration of the differences between query-based methods (e.g., our *TokenSelect*) and H_2O. Notably, *TokenSelect* leverages a dynamic selection strategy, enabling state-of-the-art performance with a minimal token budget.

2. Inability to extend sequence length: Since H_2O depends on the model's original attention mechanism, it cannot extend the effective context length. In contrast, our approach can easily extend a model with an original maximum length of 4K–32K tokens to an effective length exceeding 1M tokens.

3. Inefficient implementation: H_2O evaluates token importance based on attention scores, making it incompatible with efficient kernels such as FlashAttention (Dao, 2024). This limitation restricts its scalability. Our method, however, is designed for broad compatibility and is fully transparent to large-scale inference acceleration infrastructures, including paged attention, tensor parallelism, and prefix caching, making it ready for large-scale online serving.

To further demonstrate the superiority of *TokenSelect*, we present experimental results in Table 7. These results corroborate the findings of previous studies (Tang et al., 2024; Xiao et al., 2024a), showing that query-independent methods are inferior to query-based approaches.

Method	En.Sum	En.QA	En.MC	Math.F	R.PK	R.Num	R.KV	Avg.
H2O	2.8	0.7	0.0	6.0	2.5	2.4	0.0	2.1
InfLLM	24.3	19.5	43.7	23.7	**100.0**	**99.0**	5.0	45.0
TokenSelect	**26.9**	**21.3**	**45.8**	**28.2**	**100.0**	97.2	**48.4**	**52.5**

Table 7: Performance comparison with H_2O (Zhang et al., 2024b) on `Llama-3-8B-Instruct`, baseline performance is referenced from Xiao et al. (2024a).

F Detailed Descriptions on Baselines

In this paper, we use the following baselines:

- **NTK-Aware Scaled RoPE** (bloc97, 2023): A nonlinear RoPE interpolation method.
- **SelfExtend**: A RoPE interpolation method that reuses the position ids of neighboring tokens.
- **StreamingLLM** (Xiao et al., 2024b): The state-of-the-art method for long-context inference with predefined sparse patterns. Similar approaches include **LM-Infinite** (Han et al., 2024).
- **InfLLM** (Xiao et al., 2024a): The state-of-the-art method for long-context inference and length extrapolation using a block-level selective sparse attention method.
- **MInference** (Jiang et al., 2024): The state-of-the-art method for long-context prefilling acceleration, utilizing three sparse patterns including block-level sparse attention.
- **SnapKV** (Li et al., 2024): A fine-tuning-free approach that efficiently compresses KV caches by selecting clustered important KV positions for each attention head.
- **InfiniGen** (Lee et al., 2024): A KV cache management framework that reduces memory overhead in offloading-based LLM inference by prefetching only essential KV cache entries through selective token rehearsal.
- **QUEST** (Tang et al., 2024): A query-aware KV cache management algorithm by selecting critical KV cache based on the query-aware sparsity at page granularity.
- **RetrievalAttention** (Liu et al., 2024a): The state-of-the-art method leveraging approximate nearest neighbor search on CPU memory and an attention-aware vector search algorithm to address distribution mismatches.

G More Information on Datasets

In this paper, we use the following datasets:

- **InfiniteBench** (Zhang et al., 2024a): The mainstream long-context benchmark consisting of multi-tasks. The average length of it exceeds 200K tokens.
- **RULER** (Hsieh et al., 2024): A challenging long-context benchmark containing 13 different tasks, with subsets of varying lengths up to 128K tokens.
- **LongBench** (Bai et al., 2024): Another mainstream long-context benchmark comprising 6 types of tasks. The 95% percentile for its lengths is 31K tokens.

For InfiniteBench (Zhang et al., 2024a), we use longbook_sum_eng (En.Sum), longbook_qa_eng (En.QA), longbook_choice_eng (En.MC), longdialogue_qa_eng (En.Dia), code_debug (Code.D), math_find (Math.F), passkey (R.PK), number_string (R.Num) and kv_retrieval (R.KV) as evaluation datasets. The corresponding evaluation metrics are shown in Table 10. RULER (Hsieh et al., 2024) consists of various evaluation tasks: Single NIAH (needle in a haystack), Multi-keys NIAH, Multi-values NIAH, Multi-values NIAH, Multi-queries NIAH, Variable Tracking, Common Words Extraction, Frequent Words Extraction and Question Answering. The evaluation metric is match rate. For LongBench, we use all English tasks with evaluation metrics in Table 11.

H Comparison on Prefill Latency

We note that MInference (Jiang et al., 2024) has gained widespread adoption in real-world long-context inference applications due to its novel design of attention sparse patterns and efficient implementation based on vLLM. In the main text, we demonstrated *TokenSelect*'s performance advantages. To further prove its efficiency readiness for real-world applications, we followed Minference's

approach by comparing the end-to-end prefill latency under paged KV Cache management for different input token lengths on `Llama-3-8B` using a single A100, with results shown in Table 8. The results indicate that *TokenSelect* demonstrates significant advantages with shorter input token lengths, while maintaining efficiency comparable to MInference as input token lengths increase.

Length	FlashAttention-2 (vLLM)	MInference (vLLM)	TokenSelect
1K	0.081	3.017	0.092
10K	0.832	2.762	1.290
50K	7.717	7.540	5.712
100K	21.731	14.081	12.088
128K	32.863	18.827	15.920
200K	OOM	OOM	26.500
300K	OOM	OOM	43.406

Table 8: Comparison of end-to-end prefill latency (s).

I Detailed Performance Comparisons Under Different Cache Threshold θ

Table 9 presents the performance sensitivity to the threshold θ of the Selection Cache across various tasks. The results indicate that although θ-sensitivity varies across different task types, most tasks exhibit only slight performance degradation as θ decreases. This suggests potential for further accelerating *TokenSelect*'s decode stage by reducing θ in the vast majority of cases. It is worth noting, however, that more challenging retrieval tasks—such as R.KV—show noticeable performance degradation as θ decreases, indicating higher dynamicity requirements for token selection in these tasks.

J Experimental Results on LongBench

Compared to InfiniteBench and RULER, LongBench has much shorter text lengths. The 95% percentile for its lengths is 31K tokens. Considering that recent LLMs after SFT generally have context lengths of up to 32K tokens (Yang et al., 2024a), LongBench is less suitable for evaluating state-of-the-art long-context inference methods. Nevertheless, as shown in Table 12, our *TokenSelect* still demonstrates superior overall performance compared to most baseline methods. It's worth noting that `Yi-1.5-6B` did not yield effective results on the SAMSum task because it failed to correctly follow instructions.

K Use of AI Assistants

In this paper, AI Assistants were used for literature retrieval and grammar checking.

θ	En.Sum	En.QA	En.MC	En.Dia	Code.D	Math.F	R.PK	R.Num	R.KV	Avg.
0.5	20.99	17.83	54.31	7.50	30.20	21.14	100.00	96.10	0.20	38.69
0.6	21.21	18.08	54.31	7.50	30.20	21.36	100.00	96.78	0.20	38.84
0.7	20.73	18.08	54.31	7.50	30.46	21.36	100.00	98.98	4.40	39.53
0.8	21.47	17.85	54.31	7.50	30.20	21.58	100.00	100.00	24.60	41.94
0.85	22.39	18.15	54.31	7.50	30.20	21.79	100.00	100.00	68.20	46.94
0.9	22.62	18.86	54.31	7.50	30.20	21.71	100.00	100.00	86.60	49.08
0.95	22.46	18.54	54.31	7.50	30.56	21.77	100.00	100.00	86.20	49.05
1.0	22.66	18.68	54.31	7.50	30.51	21.78	100.00	100.00	86.84	49.15

Table 9: Performance using different selection cache similarity thresholds using `Qwen2-7B-Instruct`.

Datasets	En.Sum	En.QA	En.MC	En.Dia	Code.D	Math.F	R.PK	R.Num	R.KV
Metrics	Rouge-L-Sum	QA F1 Score	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy

Table 10: Evaluation metrics of different datasets on InfiniteBench.

Datasets	NQA	Qasper	MFQA	HQA	2WikiMQA	Musique	GovReport	QMSum
Metrics	QA F1 Score	QA F1 Score	QA F1 Score	QA F1 Score	QA F1 Score	QA F1 Score	Rouge-L	Rouge-L
Datasets	MultiNews	TREC	TQA	SAMSum	PsgCount	PsgRetrieval	LCC	RepoBench-P
Metrics	Rouge-L	Accuracy	QA F1 Score	Rouge-L	Accuracy	Accuracy	Code Sim Score	Code Sim Score

Table 11: Evaluation metrics of different datasets on LongBench.

Methods	NQA	Qasper	MFQA	HQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews
Qwen2-7B	24.24	45.42	47.79	42.76	44.38	24.16	33.80	23.78	26.17
NTK	26.25	45.94	50.76	53.20	50.31	30.83	32.75	23.21	25.94
SelfExtend	7.15	20.37	24.06	14.91	13.73	4.75	16.92	16.53	18.74
StreamLLM	19.49	42.56	39.63	42.43	44.67	15.22	31.51	20.57	26.00
InfLLM	27.47	41.44	46.99	47.47	49.29	25.62	32.68	23.10	26.77
TokenSelect	24.18	42.29	45.77	48.62	49.08	27.85	33.69	23.03	26.35
Llama-3-8B	19.85	42.36	41.03	47.38	39.20	22.96	29.94	21.45	27.51
NTK	9.90	45.35	49.41	48.86	29.22	24.56	34.31	23.82	27.27
SelfExtend	1.72	8.90	20.80	8.65	6.97	3.27	13.99	15.36	17.66
StreamLLM	20.05	42.46	39.54	43.69	37.89	19.68	29.17	21.33	27.56
InfLLM	22.64	43.70	49.03	49.04	35.61	26.06	30.76	22.70	27.57
TokenSelect	22.44	40.74	47.73	50.33	31.38	24.53	32.56	23.50	27.92
Yi-1.5-6B	17.18	32.56	39.06	36.26	39.25	16.32	30.53	20.21	26.20
NTK	0.80	35.06	29.05	7.47	24.38	0.73	13.66	6.25	25.43
SelfExtend	3.29	19.03	26.00	17.11	11.88	7.73	20.38	17.46	21.79
StreamLLM	15.05	33.27	38.31	34.91	36.92	16.33	29.38	20.02	26.14
InfLLM	17.65	36.25	45.40	41.25	35.89	16.94	30.22	20.85	26.04
TokenSelect	19.36	33.98	48.14	45.05	40.13	22.98	31.59	21.51	26.48

Methods	TREC	TQA	SAMSum	PsgCount	PsgRetrieval	LCC	RepoBench-P	**Average**
Qwen2-7B	78.50	88.77	46.33	5.50	70.00	62.40	61.95	45.37
NTK	79.50	89.51	46.03	5.50	60.00	59.36	59.69	46.17
SelfExtend	16.50	27.54	29.42	4.50	0.00	41.42	41.89	18.65
StreamLLM	75.50	87.19	46.27	3.50	27.50	61.18	61.12	40.27
InfLLM	70.50	87.51	44.53	4.00	46.50	55.08	57.53	42.90
TokenSelect	74.00	89.26	45.94	5.00	42.50	61.48	59.33	43.64
Llama-3-8B	74.00	90.50	42.30	8.50	62.50	60.83	49.14	42.46
NTK	73.00	88.74	42.51	8.87	99.50	33.62	35.04	42.12
SelfExtend	20.50	16.82	25.39	5.75	7.50	26.24	31.22	14.42
StreamLLM	73.50	90.08	41.55	5.00	49.00	60.35	48.95	40.61
InfLLM	73.50	90.91	42.43	7.17	84.00	59.88	46.48	44.46
TokenSelect	67.50	92.22	42.16	4.54	87.00	58.86	51.24	44.04
Yi-1.5-6B	71.50	48.79	0.79	3.00	28.50	57.10	52.53	32.48
NTK	40.00	12.71	1.34	0.50	3.35	54.55	37.24	18.28
SelfExtend	23.75	30.61	2.58	2.75	13.50	43.17	35.45	18.53
StreamLLM	69.00	73.36	0.82	2.50	18.50	56.37	49.05	32.49
InfLLM	71.50	71.49	1.01	4.00	10.50	56.88	46.28	33.25
TokenSelect	62.50	69.70	0.62	3.50	41.50	54.32	54.99	36.02

Table 12: Comparison of different methods with different origin models on LongBench.