TACTIC: ADAPTIVE SPARSE ATTENTION WITH CLUSTERING AND DISTRIBUTION FITTING FOR LONG-CONTEXT LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context models are essential for many applications but face inefficiencies in loading large KV caches during decoding. Prior methods enforce fixed token budgets for sparse attention, assuming a pre-chosen number of tokens can approximate full attention. However, these methods overlook variations in the sparsity of attention across heads, layers, and contexts.

To address these limitations, we propose Tactic, an adaptive and calibration-free sparse attention mechanism that dynamically selects tokens based on their cumulative attention scores rather than a fixed token budget. By setting a target fraction of total attention scores, Tactic ensures that token selection naturally adapts to variations in attention sparsity. To efficiently approximate this selection, Tactic leverages clustering-based sorting and distribution fitting, allowing it to accurately estimate token importance with minimal computational overhead.

We show that Tactic achieves superior accuracy and up to $7.29\times$ decode attention speedup, contributing to overall $1.58\times$ end-to-end inference speedup, making it a practical and effective solution for long-context LLM inference in accuracy-sensitive applications.

1 INTRODUCTION

Large language models (LLMs) power a wide range of applications, from conversational assistants to document analysis systems and search engines. The demand for multi-turn interactions and long-document processing has driven an expansion of context length, growing from thousands to as many as one million tokens (Liu et al., 2024b).

However, supporting long contexts in LLM inference presents significant challenges, primarily due to the growing memory footprint of the Key-Value (KV) cache (Tang et al., 2024). The memory requirements of the KV cache scale proportionally with the context length; therefore, it can quickly become a bottleneck despite optimizations such as Grouped-Query Attention (GQA) (Ainslie et al., 2023). Furthermore, the need to repeatedly load the KV cache for every generated token becomes a bottleneck. For instance, loading the large KV cache can account for over 50% of the total latency during auto-regressive decoding, significantly impeding the efficiency of large-scale serving systems. (Tang et al., 2024)

To mitigate the high cost of KV-cache loading, recent methods approximate full attention by selecting a subset of stored Key and Value vectors, corresponding to a subset of tokens, within a fixed token budget (Liu et al., 2024a; Tang et al., 2024; Zhang et al., 2023; Xiao et al., 2023). These approaches exploit the natural sparsity of attention, where only a small fraction of tokens significantly influence the output. By leveraging this sparsity, they aim to reduce the overhead of loading the KV-cache without sacrificing model accuracy.

Alas, existing fixed budget-based methods have several shortcomings. Some methods employ a global fixed token budget (Tang et al., 2024; Xiao et al., 2023; Zhang et al., 2023), not accounting for variations in attention sparsity across attention heads and layers. In practice, some attention heads focus on significantly more tokens than others, and the level of sparsity fluctuates across layers. More adaptive methods (Cai et al., 2024; Feng et al., 2024; Ge et al., 2024) attempt to distribute

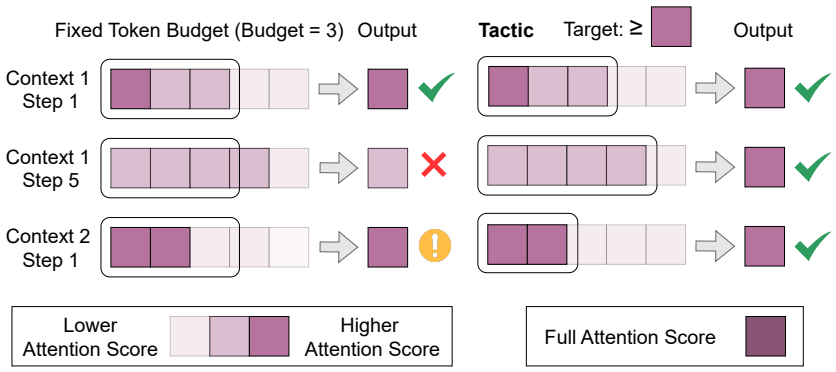


Figure 1: Comparison between fixed-budget-based methods and Tactic. Fixed-budget-based methods may select excessive tokens or have a large difference from full attention score. In contrast, Tactic dynamically selects tokens to efficiently approximate full attention based on a cumulative attention score, considering variation of sparsity across different query tokens and contexts.

token budgets more effectively using calibration data or predefined rules, but they remain constrained by static allocation and cannot adapt to query tokens and contexts, often leading to suboptimal approximations in different cases.

To address the limitations of fixed-budget-based methods, we propose Tactic, an adaptive and calibration-free post-training sparse attention mechanism that improves both the accuracy and efficiency of long-context LLM inference. Fig. 1 shows a comparison between existing fixed budget-based methods and Tactic. Instead of enforcing a fixed budget, Tactic dynamically selects tokens starting from ones with the highest attention score to ensure that their cumulative attention scores (where attention score represents the softmax output of the Query-Key product) reach a target fraction of the full attention score.

Tactic offers key advantages. First, it provides flexibility—Tactic selects fewer tokens in high-sparsity cases and more in low-sparsity cases without requiring calibration. Second, since V vectors have similar norms as we show in Fig. 12, reaching fixed cumulative attention score offers a bounded difference between sparse and full attention (see Section 2.3 and Section A).

However, efficiently selecting tokens to reach a certain threshold P of cumulative attention score is challenging. To minimize the number of tokens selected (i.e., loads from memory), the optimal way is to select tokens following a descending order of attention score until the cumulative attention score surpasses P. Additionally, unlike fixed budget-based methods that simply stop at a fixed token count, Tactic must track cumulative attention score in real time to determine the stop point, making the selection process more complex.

To approximate optimal token selection, Tactic introduces two key techniques: clustering and distribution fitting. First, to efficiently sort tokens, Tactic clusters similar tokens to reduce computational overhead. However, we observe that positional proximity, which is used for clustering tokens by prior work (Tang et al., 2024), does not necessarily guarantee similarity in Key vectors, which limits the clustering quality. Since attention operates on Query-Key interactions rather than token positions, Tactic groups tokens using K-means clustering based on Key-vector similarity (i.e., vector distance) at prefill phase. During decoding, Tactic approximates the sorted list of tokens by sorting clusters based on the similarity between Query vectors and cluster centroids. After approximating token sorting, Tactic estimates the attention score for each token by leveraging the observation that attention scores follow a smooth distribution. Using distribution fitting, Tactic effectively keeps track of attained cumulative attention score to determine the end of token selection.

By loading only the cluster centroids along with a small sampled subset of tokens ($\sim 2.5\%$ of the KV cache size in practice), Tactic efficiently selects the most critical tokens that reach the target cumulative attention score. To balance efficiency and accuracy, Tactic performs full attention on newly generated tokens and updates the clustering every fixed number of decoding steps (e.g., 2048).

Our experiments show that Tactic achieves superior and consistent accuracy compared to existing algorithms, including Quest (Tang et al., 2024), MagicPig (Chen et al., 2024), PyramidKV (Cai et al.,

2024), and Ada-KV (Feng et al., 2024), offering a more effective solution for long-context LLM inference in accuracy-sensitive applications. Tactic achieves up to 7.29× decode attention speedup, which leads to 1.58× end-to-end speedup.

In summary, we contribute the following:

- A detailed analysis of the dynamic nature of attention sparsity across heads, layers, queries, and contexts.
- **Tactic**, a sparsity-adaptive attention algorithm that uses clustering and distribution fitting to dynamically determine the token budget for achieving cumulative attention score targets.
- A comprehensive evaluation of Tactic, demonstrating Tactic consistently achieves high accuracy and significant speedup.

2 Analysis

2.1 Intrinsic Sparsity in Self-Attention Mechanisms

In the decode phase, for one request, assuming there are n previous tokens, the attention formula can be written as

$$o = \sum_{i=1}^{n} s_i v_i, \quad s_i = \frac{\exp(\frac{qk_i^\top}{\sqrt{d}})}{\sum_{i=1}^{n} \exp(\frac{qk_i^\top}{\sqrt{d}})}. \tag{1}$$

Our empirical results, evaluated on Llama 3.1 8B model using the PG19 dataset (Rae et al., 2019), show that $\|v_i\|$ are remarkably consistent, with a very low relative variance of approximately 5×10^{-5}. While in s_i, the exponential term $\exp(\frac{qk_i^\top}{\sqrt{d}})$ non-linearly amplifies the differences in the attention scores, leading to a sparse distribution (Zhang et al., 2023; Xiao et al., 2023), indicating that a small subset of tokens can exert a significant influence on the model's output. This motivates the possibility of only loading a subset of tokens to approximate the attention output and incur lower memory loading overhead.

2.2 Fixed Token Budget Approaches Lead to Accuracy Variations

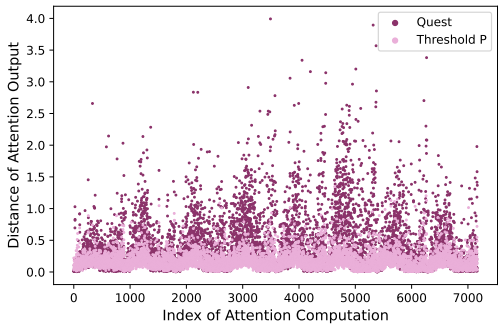


Figure 2: Distance of attention output to full attention of Quest and our proposed solution of selecting tokens until reaching cumulative attention score threshold P, measured with Llama3.1-8B-Instruct model.

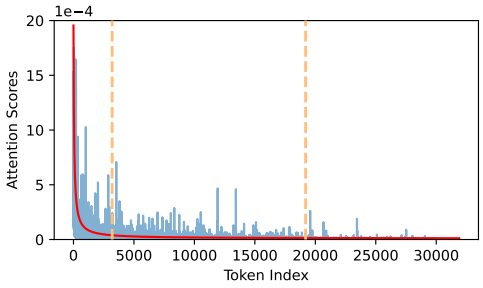
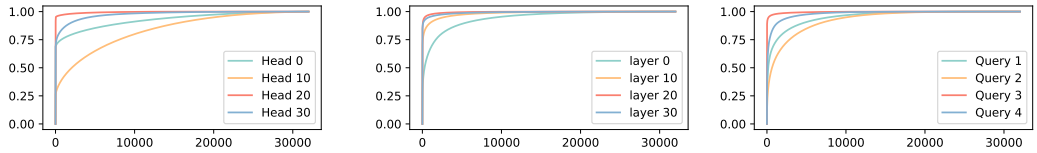


Figure 3: The distribution of attention scores after cluster-based sorting for one request in PG19 dataset using Llama3.1-8B-Instruct model. Despite some variations, the overall trend closely aligns with the function y = $\frac{a}{x} + b$.

Several methods have been proposed to choose a small set of tokens I minimizing the distance $\epsilon(I)$ between full and approximate attention. Some of the work, including Quest (Tang et al., 2024), uniformly chooses tokens across attention heads and layers. These result in a large variance of $\epsilon(I)$, as shown in Fig. 2. This variance stems from the intrinsic sparsity difference across heads



(a) (b) (c)

Figure 4: Variation in sparsity across attention heads (a), model layers (b), and query tokens (c). In (c), Query i represents the query vector of i-th decoded token. Sorted token indices (by attention score) on the x-axis and the cumulative density of attention score on the y-axis.

and layers. As illustrated in Fig. 4a, attention heads exhibit distinct sparsity patterns. Some heads display a more uniform distribution of s_i (retrieval heads), whereas others are dominated by a few high-magnitude s_i values (streaming heads). When a fixed number of tokens $|I|$ is selected per head, it leads to inefficiencies—allocating excessive tokens to streaming heads while introducing significant estimation errors in retrieval heads. Similarly, Fig. 4b highlights variation in sparsity across layers, making it inefficient to select a fixed number of tokens from different layers.

Motivated by the diversity of sparsity patterns across heads and layers, some works, including AdaKV (Feng et al., 2024) and PyramidKV (Cai et al., 2024), fix the total budget $|I|$ but use calibration data or assumptions to statically assign different budgets to different layers and heads. However, as we show in Fig. 4c, the sparsity of particular heads varies significantly depending on the query token. For example, in the model output *"The Answer is ..."*, the token *"Answer"* attends to far fewer tokens compared to *"is"*. This is because *"Answer"* relies primarily on local information to generate *"is"*, whereas *"is"* requires broader context to produce the subsequent answer. Thus, relying on static partitioning of a fixed token budget also falls short of maintaining a consistent low attention distance $\epsilon(I)$. While MagicPig (Chen et al., 2024) uses a dynamic selection, it does not provide an accuracy guarantee on $\epsilon(I)$.

2.3 Cumulative Attention Score: A More Robust Target for Sparse Attention

The key drawback of existing work is the reliance on a fixed total token budget, making it hard to adapt to sparsity variations. Instead, we propose directly using the cumulative attention score of tokens in I to guide token selection.

Specifically, we define $p(I)$ as the cumulative attention score of tokens in I, which is

$$p(I) = \sum_{i \in I} s_i = \frac{\sum_{i \in I} \exp(\frac{q k_i^\top}{\sqrt{d}})}{\sum_{i=1}^{n} \exp(\frac{q k_i^\top}{\sqrt{d}})} \tag{2}$$

These cumulative attention score targets offer two key advantages over fixed token budgets. First, they inherently adapt to sparsity variations without requiring assumptions or calibration data. Less sparse heads, layers, query tokens, and contexts naturally require more tokens to reach a given cumulative attention score than sparser ones. Second, targeting cumulative attention score provides a theoretical guarantee on attention distance. Specifically, the attention distance is bounded by

$$\epsilon(I) \leq 2(1 - p(I)) \max_i \|v_i\|. \tag{3}$$

A detailed proof is provided in Section A. Since value vectors V have similar norms across tokens (Fig. 12), setting a threshold P (typically close to 1.0) for $p(I)$ establishes a tight upper bound on $\epsilon(I)$. Identifying the minimal index set I that satisfies $p(I) \geq P$ reduces the variance of the attention approximation error, as shown in Fig. 2.

2.4 Challenges of Attaining Cumulative Attention Scores

While effective, identifying the minimal subset of tokens that achieve a target cumulative attention score during the inference is a challenging task. The optimal way is to select tokens following a

descending order of attention score until the cumulative attention score surpasses the target value. Therefore, like prior approaches, Tactic must rank tokens by attention score to minimize the number of tokens needed to reach the desired cumulative attention score. However, unlike previous methods, Tactic also requires the exact attention score values for each token to track the cumulative sum of selected tokens in real-time. This process involves two key components: (1) computing the sum of attention intermediate values, $\exp(qk^\top/\sqrt{d})$, for the selected token set I, and (2) computing the total sum of $\exp(qk^\top/\sqrt{d})$ for normalization. We discuss how Tactic achieves efficient subset identification in the following section.

3 METHODOLOGY

3.1 ALGORITHM OVERVIEW

Figure 5 provides an overview of Tactic's workflow. During prefill, Tactic performs K-means clustering on key vectors to group similar tokens. During decode, Tactic ranks tokens based on the dot product between cluster centroids and the current query vector. Tactic then models the distribution of attention score with a fitted curve and determines the tokens to meet the desired cumulative attention score threshold. After token selection, Tactic handles the Group Query Attention (GQA) and then performs the attention using FlashInfer (Ye et al., 2025).

3.2 PREFILL STAGE: GROUPING TOKENS VIA CLUSTERING

Sequence Length	WCSS (cluster)	WCSS (consecutive)
8192	173.422150	195.059021
16384	75.293724	93.293350
32768	75.394318	93.245926
65536	78.067314	93.832886

Table 1: WCSS of clustering and consecutive grouping, evaluated on Llama-3.1-8B, with data from LongBench (Bai et al., 2024).

Similar to prior works, Tactic groups tokens to reduce computational overhead for identifying critical tokens. However, existing methods rely on positional order, assuming consecutive tokens share similar attention patterns (Tang et al., 2024). However, Table 1 shows that clustering achieves lower WCSS[1] than consecutive grouping, which means consecutive grouping is suboptimal. Moreover, modern attention kernels efficiently handle non-contiguous KV-cache access, making positional grouping unnecessary. Therefore, Tactic applies K-means clustering to group tokens based on Key-vector similarity. Based on the sensitivity analysis in Section D on hyper-parameters, Tactic empirically chooses the average cluster size to be 16 to balance accuracy and efficiency and randomly samples initial cluster centroids.[2] Tactic follows the traditional K-Means algorithm with 10 iterations.

3.3 DECODE STAGE: PARTIAL SORT TOKENS USING CLUSTER CENTROIDS

Once the tokens are organized into clusters, Tactic identifies critical clusters for a given query vector Q in the decode phase. The criticality of each cluster is determined by the dot product between Q and each cluster centroid[3]. This process produces a sequence of clusters sorted by the criticality, from which we can derive a partially sorted token list.

3.4 DECODE STAGE: ESTIMATING ATTENTION SCORE VIA DISTRIBUTION FITTING

Due to the non-linearity of Softmax, cluster centroids do not accurately reflect the average attention score of individual tokens. Thus, Tactic requires a more precise approach to estimating attention

[1] $WCSS = \sum_{x \in C} \|x - \mu_c\|^2$, μ_c is the mean vector of C.

[2] Note that neither multiple initializations nor K-Means ++ initialization drastically improves the clustering quality, and in fact leads to high-performance overhead, as stated in Section D.

[3] Compared to distance, dot product directly relates to the attention score, which is more accurate.

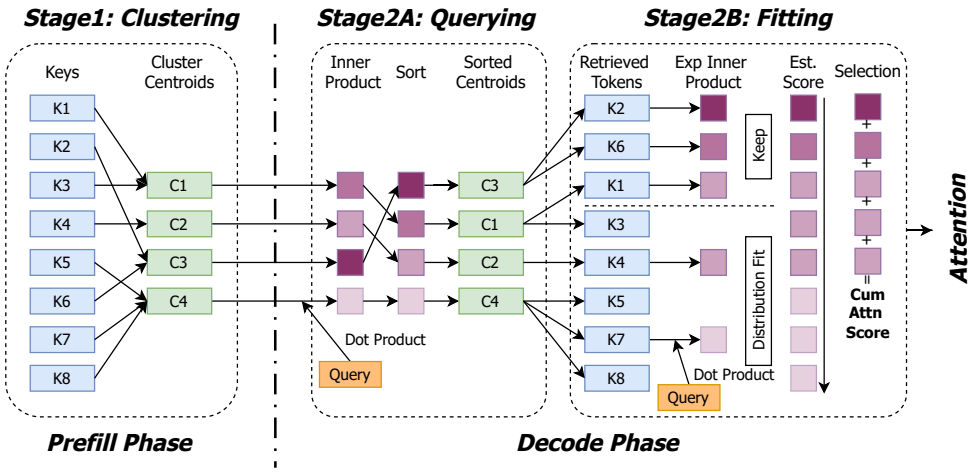


Figure 5: The overall workflow of Tactic. Tactic operates in three stages to achieve low overhead adaptive sparse attention. Stage 1 is applied right after prefill phase, Stage 2A and 2B happens during the decode phase.

score. We observe that after partial sorting, the attention score distribution follows a consistent pattern across heads, layers, and contexts. For example, as shown in Fig. 3, the attention score is high for a few tokens and then smoothly decreases, forming a long-tail distribution suggesting that function fitting can be used to estimate attention score. Tactic models the distribution of the exponential values of the dot products ($\exp(\frac{QK^\top}{\sqrt{d}})$) for each token using a lightweight function $y = \frac{a}{x} + b$, where x is the position index in the sorted list of tokens. To estimate parameters a and b, we select two segments of the tokens in the middle of the curve (e.g., 10% and 60% of all the tokens), and calculate the average of tokens within each segment (as labeled in Fig. 3). However, initial tokens (1-2% of the total tokens) are often outliers and cannot be accurately described by the curve. Moreover, these tokens feature high attention score, and thus a bad estimation would cause high deviations of estimated cumulative attention score, which affects the accuracy of Tactic. Therefore, Tactic directly calculates the exponential values of the dot products for these tokens.

3.5 Decode Stage: GQA-Aware Sparse Attention

Modern models use Grouped Query Attention (GQA) to reduce the KV cache size (Dubey et al., 2024), where multiple query heads share one KV head. However, existing methods select tokens on a per-head basis, and each query head reads KV cache independently. Tactic instead takes the union of selected tokens across all grouped query heads and loads it only once. To ensure workload balancing, Tactic divides each request into subrequests. Each subrequest processes a KV head and its corresponding Query head, with sequence length determined by the tokens selected for each KV head, allowing the request-level workload balance in modern libraries (Ye et al., 2025) to effectively handle head-level imbalance efficiently.

4 Experiments

4.1 Setting

We evaluate Tactic for both accuracy and efficiency. We use two models: Llama-3.1-8B-Instruct (Grattafiori et al., 2024), a widely used model with Grouped-Query Attention; and MegaBeam-Mistral-7B-512k (Chen Wu and Yin Song and Eden Duthie, 2024), an extended version of Mistral-7B-Instruct-v0.2 with a 512k token context window.

For accuracy evaluations, we use the PG19 language modeling dataset (Rae et al., 2019), six tasks from the LongBench dataset (Bai et al., 2024), including HotpotQA (Yang et al., 2018),

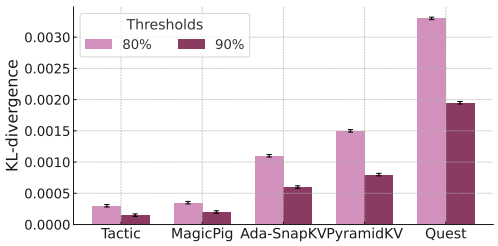


Figure 6: KL-Divergence against full attention of Tactic and other baseline methods on the PG19 dataset. Tactic maintains the most accurate output in two configurations.

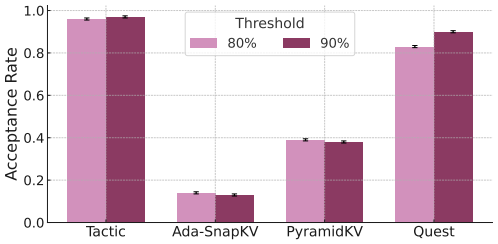


Figure 7: Acceptance rate of draft tokens at 80% and 90% thresholds on the PG19 dataset. Tactic shows more than 95% of acceptance rate, surpassing other baselines.

TriviaQA (Joshi et al., 2017), MultifieldQA (Bai et al., 2024), NarrativeQA (Kočiský et al., 2018), Qasper (Dasigi et al., 2021), and Musique (Bai et al., 2024). Additionally, we conduct experiments on the RULER benchmark (Hsieh et al., 2024), using 50 examples for each task. We compare Tactic with the most popular fixed token budget KV cache algorithms, Quest (Tang et al., 2024), PyramidKV (Cai et al., 2024), and Ada-SnapKV (Feng et al., 2024). Also, we compare Tactic with MagicPig (Chen et al., 2024) RULER benchmarks. To ensure consistency, we set the page size in Quest and the cluster size in our method to 16. Both Ada-SnapKV and PyramidKV follow the configuration settings outlined in (Feng et al., 2024), including an observation window size of 32 and a max pooling kernel size of 7. We follow the configuration of MagicPig in the original paper in RULER evaluation. For the clustering process, we limit the maximum number of iterations to 10.

For efficiency evaluations, we perform the evaluation on Nvidia Ada 6000 GPUs with CUDA 12.4 compared with full attention using Flashinfer (Ye et al., 2025).

4.2 Accuracy Evaluation

4.2.1 Output Accuracy

We assess the KL-divergence of model output probability distribution of Tactic relative to the full attention on the PG19 test set (Rae et al., 2019), under Top-K sampling. We include all texts in PG19 with the number of tokens larger than 32k. In the prefill stage, we truncate the input to 32k tokens and feed it into the model. In the decode stage, we feed tokens one by one and collect the output logits of each decode step. We collect 32 decode steps in total. As shown in Figure 6, Tactic achieves the most accurate output compared to all baselines.

4.2.2 Acceptance rate in Speculative Decoding

To further demonstrate the practical indication of smaller KL-divergence, we evaluate the token acceptance rate under greedy sampling when using Tactic as draft model for speculative decoding using the PG19 test set. Specifically, we select all documents in PG19 containing more than 32K tokens and decode up to 96 tokens per document, varying the number of draft tokens (i.e., different values of γ). During the experiments, we record the verification results, capturing the number of tokens accepted by the target model at each verification step. After computing the average number of accepted tokens for each γ, we fit a curve to the resulting data points to estimate the acceptance rate, following the formulation in Equation

$$E(\#accepted\ tokens) = \frac{\alpha - \alpha^{\gamma+1}}{1-\alpha} \tag{4}$$

which is adapted from (Leviathan et al., 2023). Here α is the acceptance rate to be estimated. The results are presented in Figure 7.

4.2.3 Accuracy for long-context tasks

LongBench. We evaluate Tactic on six LongBench tasks, as illustrated in Section 4.1. For each dataset, we first evaluate Tactic by setting the cumulative attention score threshold as 70% and 90%.

Table 2: Performance comparison on RULER for Llama-3.1-8B-Instruct and Mega-Beam-Mistral-512k models.

Llama-3.1-8B-Instruct

Methods	Config	16K	32K	64K	96K	Avg.
Full		91.3	86.0	85.2	85.0	86.8
Tactic	75%	**90.9**	**85.5**	**83.4**	**78.9**	**84.7**
PyramidKV	75%	61.8	67.4	60.8	62.5	63.1
Ada-SnapKV	75%	58.0	62.2	59.2	58.7	59.2
Quest	75%	70.0	71.5	69.7	65.7	69.2
MagicPig	75%	78.6	76.8	70.4	70.1	74.0
Tactic	90%	**90.3**	**84.9**	**82.8**	**80.5**	**84.6**
PyramidKV	90%	73.1	76.2	74.2	68.6	73.0
Ada-SnapKV	90%	72.7	76.4	74.3	68.7	73.0
Quest	90%	85.8	81.9	79.8	70.5	79.5
MagicPig	90%	79.8	76.9	71.3	70.7	74.7

Mega-Beam-Mistral-512k

Methods	Config	16K	32K	64K	96K	Avg.
Full		90.9	88.4	82.7	83.1	86.3
Tactic	75%	88.0	**88.8**	**81.7**	**82.5**	**85.2**
PyramidKV	75%	80.2	79.0	75.2	74.0	77.1
Ada-SnapKV	75%	80.6	78.1	75.4	73.6	76.8
Quest	75%	80.7	79.0	71.4	70.5	75.4
MagicPig	75%	**89.7**	86.5	81.0	69.4	81.6
Tactic	90%	**90.3**	**88.0**	**81.0**	82.6	**85.4**
PyramidKV	90%	84.3	82.7	76.2	86.2	82.4
Ada-SnapKV	90%	84.7	81.8	76.2	**87.1**	82.4
Quest	90%	81.1	81.3	73.5	79.7	78.9
MagicPig	90%	90.2	87.1	82.3	82.3	85.4

The average number of tokens selected at each threshold serves as the token budget for evaluating baselines. As shown in Table 5, Tactic consistently outperforms all other baselines. At a threshold of 90%, Tactic achieves performance close to full attention.

RULER. We evaluate Tactic and baselines on all tasks in RULER (Hsieh et al., 2024) with context length ranging from 16K to 96K. As shown in the Table 2, Tactic consistently outperforms all baselines in each configuration in terms of average accuracy. Furthermore, at higher thresholds, Tactic achieves similar accuracy to full attention, significantly higher than other methods. Also, we provide the efficiency comparison in the RULER test in Section G.

We provide a detailed table of the average number of tokens selected by Tactic across various thresholds, datasets, and models in Table 6, Table 7, and Table 8, which is set as token budgets for baselines.

4.2.4 Accuracy of Clustering & Distribution Fitting

To identify the minimal number of tokens to reach the threshold, Tactic employs clustering and distribution fitting (explained in Section 3). We evaluate our method on the PG19 dataset, focusing on how well it aligns with the target cumulative attention score and how many tokens it selects. We set specific attention score thresholds and compare the actual cumulative score achieved by our method against two oracles: the global optimal, which sums tokens in the descending order of attention score, and the clustering optimal, which sums attention score from sorted clusters.

Table 3 indicates that Tactic achieves the target threshold of cumulative attention score with high success rates. Also, the values of *Cluster Optimal* and *Tactic* are close, indicating that the distribution fitting presents an accurate estimation of the number of tokens.

Table 3: Evaluation of number of tokens selected and ratio of cumulative attention score achieved for llama model

Threshold	Optimal	Cluster Optimal	Tactic	Average Achieved Score	Success Rate
50%	71	166	185	66%	92%
60%	122	271	294	72%	89%
70%	212	451	490	78%	86%
80%	394	802	890	84%	84%
90%	895	1723	1975	91%	86%

4.3 Efficiency Evaluation

We begin by analyzing the latency breakdown of Tactic, focusing on token clustering during the prefill phase and attention computation for critical tokens during decoding (Section 4.3.1). Next, we evaluate Tactic's end-to-end performance and its speed-up relative to full attention (Section 4.3.3).

4.3.1 Latency Breakdown

Latency of clustering during prefill. We measure the time taken for clustering for different sequence lengths in Figure 8. We observe that, as the sequence length increases, the clustering time increases

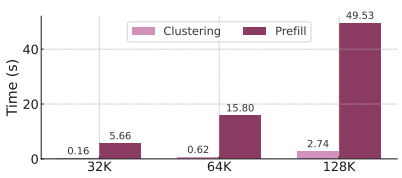


Figure 8: Comparison of Clustering and Prefill time across different sequence lengths.

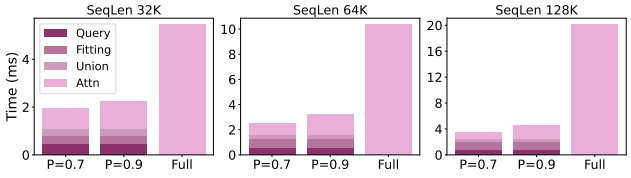


Figure 9: Latency breakdown of Tactic in the decode stage for different sequence lengths and thresholds.

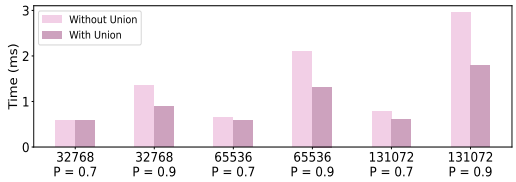


Figure 10: Ablation study on taking union for the GQA model. Taking union significantly reduces the attention time.

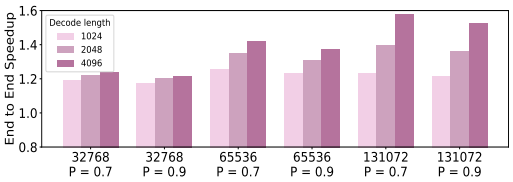


Figure 11: End-to-end speedup of Tactic compared to the full attention.

quadratically and is dominated by the distance calculation. However, long sequences also significantly increase the prefill time. Overall, the clustering time always stays below 6% of the prefill time.

Latency of attention during decode. In the decode stage, Tactic identifies and performs attention on critical tokens. We break down this process into four parts: 1) Cluster sorting, where the clusters are ranked based on the dot product of centroids and queries, 2) Distribution fitting, where Tactic samples a small portion of tokens and derives the attention score to identify the token budget for each attention head, 3) performing attention for the selected tokens. Figure 9 shows the latency of this breakdown for different sequence lengths.

The latency of sparse attention during decode is reduced significantly, while the overheads of sorting and distribution fitting remain low across various sequence lengths. Overall, Tactic achieves up to $7.29\times$ speedup compared to the full attention.

4.3.2 Ablation Study for Query Head Union

We evaluate the benefits of taking the union of grouped query heads versus computing attention for each query head individually. As shown in Fig. 10, across different context lengths and ratio P, taking unions can achieve up to $1.65\times$ attention speedup, due to the reduced memory loading.

4.3.3 End-to-end performance

We compute the end-to-end performance of Tactic with different output tokens, sequence lengths, and ratios in Figure 11, considering the prefill stages and the clustering overhead. Overall, Tactic achieves a speedup of up to $1.58\times$ compared to full attention.

5 Conclusion

We presented Tactic, a sparsity-adaptive attention mechanism for efficient long-context LLM inference. Unlike fixed token budget methods, Tactic dynamically selects tokens based on cumulative attention scores, adapting to variations in attention sparsity. By leveraging clustering-based sorting and distribution fitting, Tactic accurately estimates token importance with low overhead. Our results showed that Tactic outperforms existing sparse attention methods, achieving higher accuracy and significant inference speedups, making it a practical solution for long-context LLMs.

REFERENCES

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

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling, 2024. URL https://arxiv.org/abs/2406.02069.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. Magicpig: Lsh sampling for efficient llm generation, 2024. URL https://arxiv.org/abs/2410.16179.

Chen Wu and Yin Song and Eden Duthie. aws-prototyping/MegaBeam-Mistral-7B-512k, 2024. URL https://huggingface.co/aws-prototyping/MegaBeam-Mistral-7B-512k.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference, 2024. URL https://arxiv.org/abs/2407.11550.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms, 2024. URL https://arxiv.org/abs/2310.01801.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

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

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. doi: 10.1162/tacl_a_00023. URL https://aclanthology.org/Q18-1023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression, 2024a. URL https://arxiv.org/abs/2412.03213.

Xiaoran Liu, Hang Yan, Shuo Zhang, Chenxin An, Xipeng Qiu, and Dahua Lin. Scaling laws of rope-based extrapolation, 2024b. URL https://arxiv.org/abs/2310.05209.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019. URL https://arxiv.org/abs/1911.05507.

Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL https://www.sciencedirect.com/science/article/pii/0377042787901257.

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

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv*, 2023.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259.

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

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

A Bound of approximation error

In Section 2.3, the upper bound of $\epsilon(I)$ can be derived as

$$\begin{aligned}
\epsilon(I) =& \|o - \tilde{o}(I)\| && (5)\\
=& \|o - \frac{1}{p(I)} \sum_{i \in I} s_i v_i\| && (6)\\
=& \|\sum_{i \in I} s_i v_i + \sum_{i \notin I} s_i v_i - \frac{1}{p(I)} \sum_{i \in I} s_i v_i\| && (7)\\
=& \|\left(1 - \frac{1}{p(I)}\right) \sum_{i \in I} s_i v_i + \sum_{i \notin I} s_i v_i\| && (8)\\
\leq& \|\left(1 - \frac{1}{p(I)}\right) \sum_{i \in I} s_i v_i\| + \|\sum_{i \notin I} s_i v_i\| && (9)\\
\leq& \left|\left(1 - \frac{1}{p(I)}\right)\right| \sum_{i \in I} s_i \|v_i\| + \sum_{i \notin I} s_i \|v_i\| && (10)\\
\leq& \left(\frac{1}{p(I)} - 1\right) p(I) \max_i \|v_i\| + (1 - p(I)) \max_i \|v_i\|. && (11)
\end{aligned}$$

Hence we can get

$$\epsilon(I) \leq 2(1 - p(I)) \max_i \|v_i\|. \quad (12)$$

Both (10) to (11) and (11) to (12) are based on triangle inequality.

B Value Vector Norm Distribution

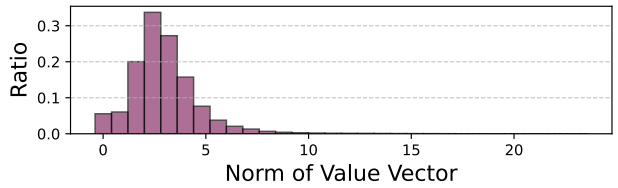


Figure 12: The distribution of $||V||$ across different layers, heads, and decoding tokens. The results indicate that $||V||$ values are concentrated within a very narrow range.

C Algorithm of Distribution Fitting

The algorithm for distribution fitting is illustrated in Algorithm 1.

Algorithm 1 Estimating Token Budget via Distribution Fitting

Input: Key sequence unpacking from sorted clusters $\{k_1, k_2, \ldots, k_n\}$, query Q, weight percentage threshold P, initial token count N, centers of two sampling window p_1 and p_2, sampling window size w, head dimension d
Output: Token budget K

Compute μ_1 and μ_2 as the means of $\exp(k_i \cdot Q/\sqrt{d})$ within fixed windows around p_1 and p_2, namely,

$$\mu_j = \frac{\sum_{i=p_j-\frac{w}{2}}^{p_j+\frac{w}{2}} \exp(k_i \cdot Q/\sqrt{d})}{w},\ j = 1, 2 \tag{13}$$

Solve for parameters a and b in $y = a/x + b$ using (p_1, μ_1) and (p_2, μ_2).

Initialize array $s_i,\ i \in [n]$ to store the fitted attention scores for all tokens
for $i = 1$ to n **do**
    If $i \leq N$, $s_i = \exp(x_i \cdot Q/\sqrt{d})$
    Else, $s_i = a/i + b$
end for
Compute the minimal K such that the cumulative sum $\sum_1^K s_i \geq P \cdot \sum_1^n s_i$.

return K

D Sensitivity Analysis of Clustering

As stated in Section 3.2, clustering is controlled by three parameters: the number of iterations, the cluster size, and the initialization count (n_init). In this section we provide the effects of different value of the three parameters. We use Silhouette Score (Rousseeuw, 1987) to evaluate the performance of clustering, which is a commonly used metric in machine learning field.

Table 4 presents the impact of key hyperparameters on clustering quality across different sequence lengths. Increasing the number of iterations or number of initializations brings little improvement on the performance of clustering but proportional overhead, Tactic chooses 10 iterations and one time initialization to maintain high performance with minimal overhead. Also, a smaller cluster size yields better clustering performance but longer clustering time, Tactic chooses 16 to balance the performance and efficiency. Notably, the effect of these hyperparameters is largely *independent of sequence length*.

Hyperparameter	Value	8192	16384	32768	65536
n_init	1	0.091	0.113	0.145	0.166
	2	0.095	0.117	0.150	0.170
	4	0.095	0.117	0.149	0.170
	8	0.095	0.118	0.149	0.170
max_iter	5	0.091	0.111	0.145	0.166
	10	0.091	0.113	0.144	0.165
	20	0.095	0.117	0.150	0.171
	30	0.095	0.117	0.150	0.171
	40	0.095	0.117	0.150	0.171
	50	0.095	0.117	0.150	0.171
cluster_size	8	0.096	0.118	0.155	0.173
	16	0.095	0.116	0.149	0.170
	32	0.088	0.110	0.143	0.164
	64	0.082	0.104	0.131	0.155
	128	0.070	0.096	0.119	0.143

Table 4: Silhouette scores for K-Means clustering across different sequence lengths, evaluated against hyperparameters **n_init**, **max_iter**, and **cluster_size**.

E LongBench accuracy evaluation

The LongBench evaluation results are presented in Table 5. MagicPig is not included since they don't provide an evaluation script on Longbench.

Methods	Config	HotpotQA	TriviaQA	MultiFieldQA	Qasper	NarrativeQA	Musique
Llama-3.1-8B-Instruct	Full	54.99	89.22	55.05	46.52	27.12	32.16
Tactic	70%	51.20	**90.38**	**52.98**	**43.30**	**30.39**	**28.57**
PyramidKV	70%	**52.59**	89.57	43.26	27.50	21.44	25.34
AdaKV	70%	49.32	89.57	43.45	29.99	25.23	23.54
Quest	70%	45.43	77.42	49.96	38.09	24.77	24.78
Tactic	90%	53.57	**90.61**	**54.35**	**44.20**	29.59	**30.71**
PyramidKV	90%	53.77	90.31	48.15	36.40	26.97	28.52
AdaKV	90%	**54.05**	90.46	49.15	37.55	27.86	29.19
Quest	90%	49.37	80.38	52.42	42.41	**30.21**	26.64
MegaBeam-Mistral-512k	Full	48.89	88.24	52.14	33.13	26.08	26.38
Tactic	70%	**49.15**	87.89	50.50	**32.37**	**25.63**	**25.85**
PyramidKV	70%	42.21	85.77	36.74	21.23	19.31	19.93
AdaKV	70%	42.23	85.65	38.44	22.23	21.89	21.68
Quest	70%	48.90	**88.13**	**50.58**	30.78	23.88	24.65
Tactic	90%	49.59	**89.16**	49.85	**33.93**	**26.31**	**25.93**
PyramidKV	90%	44.05	86.64	40.66	24.22	21.13	23.32
AdaKV	90%	44.80	86.80	42.80	22.51	22.46	24.86
Quest	90%	**51.69**	88.49	**51.81**	32.46	24.63	25.89

Table 5: Experiment Results on LongBench

F Number of tokens selected by Tactic in Benchmarks

We provide the average number of tokens selected by Tactic in benchmark evaluations in Table 6, Table 7 and Table 8.

G Speedup of Tactic in RULER evaluation

To ensure a fair comparison of efficiency between Tactic and baseline methods in Section Section 4.2.3, we adjust the number of selected tokens to match the RULER scores. The token count required to achieve this accuracy, along with the attention speedup over baseline methods, is shown in Table 9.

Table 6: Average number of tokens selected by Tactic for different cumulative attention scores.

P	HotpotQA	TriviaQA	MultiFieldQA	Qasper	NarrativeQA	Musique
Llama-3.1-8B-Instruct						
70%	959	761	774	629	1918	1229
90%	2298	1813	1559	1276	4254	2754
MegaBeam-Mistral-7B-512k						
70%	2126	1733	1399	1191	3017	2598
90%	3641	3048	2031	1546	5616	4384

Table 7: Average number of tokens selected by Tactic for Llama-3.1-8B-Instruct across context lengths and cumulative attention scores.

Task	16K		32K		64K		96K	
	75%	90%	75%	90%	75%	90%	75%	90%
NIAH_Single 1	166	813	363	1567	456	2404	1790	3319
NIAH_Single 2	271	1289	534	2196	711	3209	2030	3940
NIAH_Single 3	171	1015	369	1820	507	3031	1068	3952
NIAH_Multikey 1	224	1052	449	1832	654	2792	978	4438
NIAH_Multikey 2	399	1612	706	2533	981	3902	1405	5527
NIAH_Multikey 3	340	1428	567	2404	798	3990	1068	5062
NIAH_Multivalue	246	1769	478	2679	692	4458	1025	8147
NIAH_Multiquery	253	1524	465	2648	681	4830	915	6011
FWE	282	1572	515	2693	778	4376	963	6202
CWE	443	1939	570	3036	655	4707	780	6731
QA 1	329	1250	704	2565	852	3830	3417	5055
QA 2	547	1484	1092	2263	1662	3455	2120	4276
VT	118	731	253	1413	269	2028	404	3068
AVG	291	1344	543	2281	746	3616	1382	5056
AVG/Token	1.78%	8.21%	1.66%	6.96%	1.14%	5.52%	1.41%	5.14%

Table 8: Average number of tokens selected by Tactic for MegaBeam-Mistral-7B-512K across context lengths and cumulative attention scores.

Task	16K		32K		64K		96K	
	75%	90%	75%	90%	75%	90%	75%	90%
NIAH_Single 1	1410	1176	2521	2354	4715	4512	7042	7271
NIAH_Single 2	1418	1665	2704	3364	5113	6732	7668	10472
NIAH_Single 3	3005	2032	5312	3800	10235	7327	15492	11090
NIAH_Multikey 1	1486	1661	2872	3415	5332	5983	7781	8767
NIAH_Multikey 2	1480	1639	2826	3031	5288	5956	8240	9279
NIAH_Multikey 3	2840	2039	5990	4306	11438	7741	17088	11292
NIAH_Multivalue	2880	2225	5070	4006	10180	7463	16138	11853
NIAH_Multiquery	3088	2165	5234	4075	9931	7551	14420	11164
FWE	2706	1809	5537	3685	10837	7157	16558	10818
CWE	4240	2526	7404	4869	13189	8802	18329	13215
QA 1	1003	1502	2772	3132	5828	5685	6007	9150
QA 2	724	1718	2124	3017	4853	6520	8598	8598
VT	2227	1409	4132	2619	8667	4719	14318	8671
AVG	2193	1813	4192	3513	8124	6627	12129	10126
AVG/Token	13.38%	11.06%	12.79%	10.72%	12.40%	10.11%	12.34%	10.30%

Table 9: Attention speedup over baselines in RULER evaluation.

Method	SeqLen	# Tokens Chosen	Attention Time (ms)
Tactic	32K	543 (1×)	0.58 (1×)
Quest	32K	5430 (10×)	1.08 (1.86×)
Pyramid	32K	8145 (15×)	1.47 (2.53×)
Ada	32K	8688 (16×)	1.56 (2.69×)
Tactic	64K	746 (1×)	0.57 (1×)
Quest	64K	9698 (13×)	1.69 (2.96×)
Pyramid	64K	16412 (22×)	2.67 (4.68×)
Ada	64K	14920 (20×)	2.45 (4.29×)
Tactic	128K	1381 (1×)	0.66 (1×)
Quest	128K	20715 (15×)	3.31 (5.01×)
Pyramid	128K	34525 (25×)	5.43 (8.22×)
Ada	128K	33144 (24×)	5.23 (7.92×)

H Computer information of experiments

For Longbench and RULER evaluation (Table 5 and Table 2), we use a Nvidia H100 DGX server, Longbench evaluation takes around 8 hours and RULER evaluation takes 12 hours, two models (Llama-3.1-8B-Instruct and MegaBeam-Mistral-7B-512K). Other accuracy evaluations like KL-divergence can be done within 30 minutes on a single H100. For Efficiency evaluation, we use Nvidia Ada 6000 GPUs with CUDA 12.4.

I Potential Social Impacts

This paper is motivated by recent advances in the field of long-context language models. Sparse attention methods has the potential to be used to serve more requests and thus benefit more users.