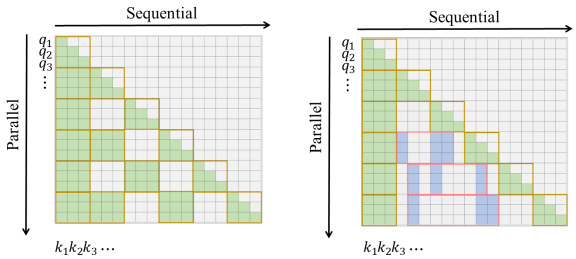
AnchorAttention: Difference-Aware Sparse Attention with Stripe Granularity

Yu Zhang[1,*], **Dong Guo**[3,*], **Fang Wu**[1],
Guoliang Zhu[4], **Dian Ding**[2,†], **Yiming Zhang**[2,1,†]
[1]Xiamen University, [2]Shanghai Jiao Tong University, [3]Xi'an Jiaotong University, [4]GMICloud

Abstract

Large Language Models (LLMs) with extended context lengths face significant computational challenges during the pre-filling phase, primarily due to the quadratic complexity of self-attention. Existing methods typically employ dynamic pattern matching and block-sparse low-level implementations. However, their reliance on local information for pattern identification fails to capture global contexts, and the coarse granularity of blocks leads to persistent internal sparsity, resulting in suboptimal accuracy and efficiency. To address these limitations, we propose **AnchorAttention**, a difference-aware, dynamic sparse attention mechanism that efficiently identifies critical attention regions at a finer stripe granularity while adapting to global contextual information, achieving superior speed and accuracy. AnchorAttention comprises three key components: (1) **Pattern-based Anchor Computation**, leveraging the commonalities present across all inputs to rapidly compute a set of near-maximum scores as the anchor; (2) **Difference-aware Stripe Sparsity Identification**, performing difference-aware comparisons with the anchor to quickly obtain discrete coordinates of significant regions in a stripe-like sparsity pattern; (3) **Fine-grained Sparse Computation**, replacing the traditional contiguous KV block loading approach with simultaneous discrete KV position loading to maximize sparsity rates while preserving full hardware computational potential. With its finer-grained sparsity strategy, **AnchorAttention** achieves higher sparsity rates at the same recall level, significantly reducing computation time. Compared to previous state-of-the-art methods, at a text length of 128k, it achieves a speedup of 1.44× while maintaining higher recall rates.



(a) Block sparse (b) Stripe sparse

Figure 1: (a) Block-sparse pattern, with yellow regions indicating computed blocks; (b) Stripe-sparse pattern, with red regions showing computed areas, enabling higher sparsity by loading non-contiguous positions across multiple blocks.

1 Introduction

Large Language Models (LLMs) have brought transformative advancements to numerous domains by enabling sophisticated natural language understanding and generation(Zhou et al., 2024; Kaddour et al., 2023; Qin et al., 2024). However, as the supported context lengths continue to increase, the inference cost — particularly in the prefill phase — has become a major bottleneck. This is primarily due to the quadratic computational complexity of full-attention mechanisms with respect to sequence length, which leads to significant efficiency issues in long-sequence inference tasks.

To mitigate the computational overhead during the prefill phase, FlashAttention (Dao et al., 2022) leverages memory transfer disparities across hardware hierarchies and incorporates the online Softmax algorithm (Milakov and Gimelshein, 2018), thereby significantly reducing transmission costs at the hardware level. Meanwhile, several studies (Li et al., 2024; Zhang et al., 2023; Fu et al., 2024; Yang et al., 2024) have revealed the inherent sparsity in attention mechanisms, demonstrating that retaining only a small subset of key-value (KV) pairs is sufficient to preserve model accuracy. How-

*Equal contribution
†Corresponding author

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 8549–8561
November 4-9, 2025

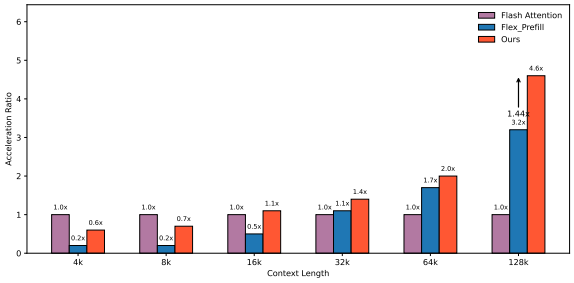


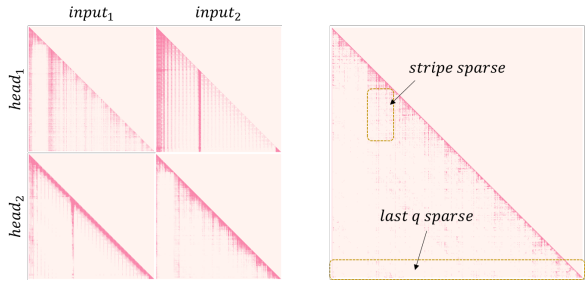
Figure 2: Acceleration of attention computation compared to FlashAttention, where our method provides significant speedup under a 128k context length.

ever, these methods still rely on computing full attention scores to identify the retained KV subset and therefore do not reduce the runtime cost during the prefill phase. Recent efforts have attempted to exploit sparsity to optimize prefill computation. For example, StreamingLLM (Xiao et al., 2024) introduces a sparse pattern that retains only local and initial positions during computation, significantly accelerating attention, but often missing essential information from intermediate content. Minference (Jiang et al., 2024) proposes that attention patterns follow multiple sparse modes and accelerate computation by applying offline-searched sparse configurations. However, its static design cannot adapt to diverse input patterns and often fails to select the optimal mode configuration. FlexPrefill (Lai et al., 2025) improves upon this by dynamically selecting patterns online, yet its selection heavily depends on local information, limiting its generality. SpargeAttn (Zhang et al., 2025) and X-Attention (Xu et al., 2025) attempt to identify informative blocks using similarity-based or diagonal priors. However, these designs primarily target general-purpose models and lack specialized mechanisms for language model characteristics, particularly in exploiting architectural properties like the *attention sink*(Xiao et al., 2024) phenomenon. On the other hand, as shown in Fig. 1a, existing methods typically rely on coarse-grained block-level KV selection in attention computation, which is misaligned with the naturally fine-grained sparsity observed in attention maps (see Fig. 3b), inevitably leading to redundant attention computations.

To address these challenges, we propose **AnchorAttention**, a difference-aware sparse attention strategy with stripe granularity. AnchorAttention introduces a sparsity mechanism centered around the concept of an **anchor**, inspired by the common structural patterns observed in attention distributions across all inputs. We observe that the maximum values after dot-product computations consistently emerge at initial or local window positions. We therefore extract the maximum score from these regions and designate it as the **anchor**. The importance of other positions is then determined by directly comparing their values against the anchor, effectively bypassing expensive sorting operations. In contrast to traditional block-level sparsity methods (Zhang et al., 2025; Xu et al., 2025; Lai et al., 2025; Jiang et al., 2024; Yang et al., 2025), AnchorAttention adopts a more flexible *stripe-level sparsity* strategy, reducing the identification granularity from coarse blocks to finer-grained stripes and enabling higher sparsity rates. During sparse computation, we maintain the same computation flow as FlashAttention(Dao et al., 2022), with the primary modification being the replacement of the contiguous KV loading scheme with a discrete KV loading approach. Compared to block-sparsity-based strategies, this enables us to enhance recognition precision while preserving parallel computation efficiency. **AnchorAttention** comprises the following three steps:**Pattern-based Anchor Computation**: We observe that the distribution of the most significant values remains fixed and stable across various input transformations. We first compute these values and designate the obtained approximate maximum value as the anchor. **Difference-aware Stripe Sparsity Identification**: Compared to block sparsity, we adopt a finer-grained stripe sparsity approach. By performing dot-product computations between the compressed query and the full set of keys, we use direct comparisons with the anchor's difference to rapidly identify which keys and values are significant, avoiding costly sorting operations. **Fine-grained Sparse Computation**: We transition from block sparsity's continuous KV loading to discrete KV loading. During computation, we maintain block-based computations to maximize sparsity while preserving parallel computing capabilities.

We evaluate **AnchorAttention** on Llama-3.1-8B-Instruct (Touvron et al., 2023) and Qwen2.5-7B-Instruct(Qwen et al., 2025) across various context lengths. The benchmarks used include RULER (Hsieh et al., 2024), Needle In A Haystack (Kamradt, 2023), and Longbench (Bai et al., 2024). All of our experiments are conducted under context lengths up to 128k. Our goal is not

to endlessly extend the context length while relying on simple-task performance as the evaluation metric but rather to approximate full attention with minimal computation. Therefore, we adopt recall as the primary evaluation metric. Under this criterion, our method surpasses the state-of-the-art FlexPrefill (Lai et al., 2025) in recall while achieving a 1.44× speedup. Compared to full KV FlashAttention (Dao et al., 2022), our method achieves a 4.6× speedup, significantly reducing attention computation time. The results demonstrate that **AnchorAttention** delivers substantial acceleration while preserving model accuracy. Under this criterion, our method surpasses the state-of-the-art FlexPrefill (Lai et al., 2025) in recall while achieving a 1.44× speedup. Compared to full KV FlashAttention (Dao et al., 2022), our method achieves a 4.6× speedup without sacrificing accuracy, as shown in Fig. 2, significantly reducing attention computation time. These results demonstrate that **AnchorAttention** delivers substantial acceleration while preserving model accuracy.



(a) Heatmaps of different inputs

(b) Stripe sparse and local information sparse

Figure 3: (a) Heatmaps vary significantly across different inputs. (b) Stripe sparse appears in specific attention maps, demonstrating that local information fails to capture the full attention distribution.

2 Analysis and Observation

2.1 Analysis

In this section, we primarily analyze the impact of identification schemes and identification granularities on the final recall rate, elucidating how different identification approaches and granularities affect the output.

2.1.1 Performance of Different Identification Schemes in Sparsity Strategies

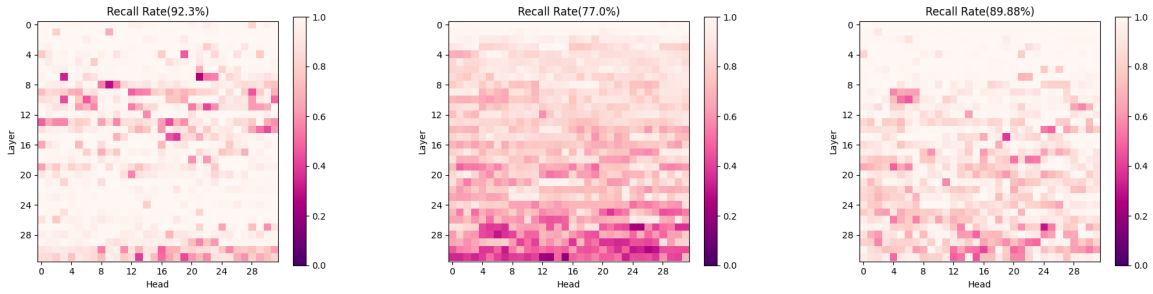
Previous work has widely adopted top-k(Xiao et al., 2024; Li et al., 2024; Holmes et al., 2024; Tang et al., 2024; Liu et al., 2024a) and top-cdf(Lai et al., 2025) strategies to identify important positions in sparsity strategies. In the top-k strategy, the value of k is fixed. As shown in Figure 4a, this static selection of k can result in some heads having recall rates well below the target, prompting prior methods to assign different k values for different heads. However, such static k settings often perform poorly with dynamic inputs, as further detailed in Appendix B. To address this limitation, some methods employ the top-cdf strategy (see Figure 4b), which ensures each head meets the desired recall rate by computing cumulative attention scores. However, both approaches rely on sorting, incurring significant computational overhead. In contrast, the difference-aware strategy (see Figure 4c) begins with a known maximum value and directly subtracts other values to obtain the differences. If the difference exceeds a predefined threshold, subsequent computations are skipped. This method eliminates the need for sorting operations and achieves performance comparable to that of top-cdf, while the maximum value, as discussed in Section 2.2.2, can be obtained with minimal computational overhead.

Method	Recall Rate	Sparsity Rate
Block (Top-K=256)	88.5%	56.3%
Stripe (Top-K=16384)	91.2%	76.6%

Table 1: Comparison of block and stripe granularity in sparsity strategies for LLaMA-3.1-8B-Instruct on the 128k Ruler(Hsieh et al., 2024) dataset, where the stripe granularity achieves higher recall at higher sparsity rates.

2.1.2 Performance of Different Identification Granularities in Sparsity Strategies

In Section 2.1.1, we systematically analyze the impact of various strategies on the final recall rate. However, identifying these positions typically requires computing full attention scores, which provides only limited acceleration for attention computation itself. Many existing methods rely on underlying block-sparse attention implementations and adopt different block identification schemes. Yet, as discussed in Section 2.2.1, not all elements within a block are equally important; the heatmap often exhibits stripe-shaped sparsity. To alleviate the overly coarse block granularity, we simplify the block size to retain only the column dimension and set the row dimension to 1, which we term "stripe granularity."



(a) Top-K(4096) (b) Top-CDF(0.95) (c) Difference-Aware(11)

Figure 4: Recall heatmaps of sparsity strategies using LLaMA-3.1-8B-Instruct on the 128k Ruler(Hsieh et al., 2024) dataset, with average sparsity rates of 93.7% (a), 96.4% (b), and 94.1% (c), and corresponding recall rates of 92.3% (a), 77.0% (b), and 89.88% (c). Additional heatmap analyses for other inputs are provided in Appendix A, where the numbers in parentheses indicate the parameter choices. These results demonstrate the limitations of the top-k strategy across different inputs, while the difference-based comparison achieves performance consistent with top-cdf. Recall is defined as the percentage of attention values that are numerically equal between the current sparse attention and the full attention(Jiang et al., 2024).

Through a comparison between the stripe-granularity strategy and traditional block-sparse identification (block granularity (128,128) vs. stripe granularity (128,1)), we evaluate the achievable sparsity rates under equivalent recall thresholds. As shown in Table 1, the stripe-granularity approach attains higher sparsity rates at comparable or higher recall targets. Moreover, from an implementation perspective in Triton, the inner loop with discrete KV loading only requires the corresponding indices; on widely used hardware such as the A100, the computational overhead is nearly identical at the same sparsity level. These results provide an implementation-level, stripe-based sparsification alternative to traditional block-sparse methods.

2.2 Observation

In this section, we primarily discuss our observations of attention patterns, where sparsity exhibits broad dynamism while simultaneously maintaining consistent static properties.

2.2.1 Diversity of Sparse Attention Patterns

Sparse attention patterns are prevalent in large language models, yet the sparsity distribution within a single attention head varies significantly due to input content(Lai et al., 2025). As shown in Figure 3a, different inputs yield distinct sparsity patterns, indicating that static pattern recognition cannot adapt to dynamic inputs, **necessitating more flexible sparsity strategies**. Additionally, Figure 3b shows that critical information often appears at a finer granularity, concentrating in only a few columns and forming a striped pattern in the heatmap. This phenomenon highlights that using block sparsity as the minimum granularity fails to fully leverage sparsity, **underscoring the need for finer-grained selection strategies**.

Moreover, Figure 3b demonstrates that relying solely on the local information from the last query fails to reconstruct the full attention heatmap(Jiang et al., 2024; Lai et al., 2025), as these stripes may vanish at the end, highlighting that local information lacks generalizability and **requiring broader positional data**.

2.2.2 Commonality of Sparse Attention Patterns

Although sparsity patterns vary significantly across different models, certain consistent features remain prominent. As shown in Figures 3a and 3b, the attention scores **at the local window positions and the initial token position are consistently critical**. We further analyze these positions in Figure 5, examining the first token and a local window of 128 tokens under a 128k context length. The results show that in the LLaMA (Touvron et al., 2023) model, approximately 99% of the highest attention scores are concentrated in these regions, whereas in the Qwen (Qwen et al., 2025) model, the proportion is around 90%. Although prior works (Xiao et al., 2024; Jiang et al., 2024) have identified the importance of these positions and focused on preserving them, their potential for guiding the construction of broader sparsity structures remains underexplored.

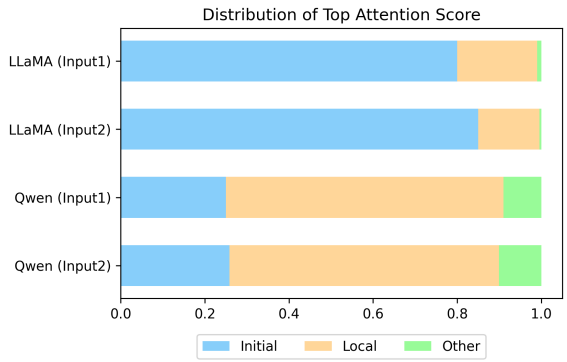


Figure 5: The distribution of maximum attention scores highlights the dominance of anchor positions.

In contrast, we propose to define these high-impact positions as **anchor**, emphasizing their critical role in attention computation and their utility in precomputing and approximating the sparsity distribution of other positions.

3 Method

In this section, we present **AnchorAttention**, a **difference-aware** and **stripe** sparse attention strategy. **AnchorAttention** consists of three key components: (1) **Pattern-based Anchor Computation**, (2) **Difference-aware Stripe Sparsity Identification**, and (3) **Fine-Grained Sparse Computation**. We implement all three strategies as kernel operations, as described in (4) **Kernel Optimization and Algorithm**.

3.1 Pattern-based Anchor Computation

As discussed in Section 2.2.2, attention scores consistently exhibit prominent peaks in two specific regions: the initial token positions and the local window position. This structurally stable pattern motivates us to explicitly compute the attention scores at these positions and define the resulting maximum value as the **anchor**.

The anchor computation is highly efficient, as it requires only a small subset of the key. This enables us to approximate the maximum attention score at a very low computational cost, avoiding the need to compute the full attention. The computed anchor can then directly guide the selection of sparse attention patterns.

Formally, the anchor is computed as follows:

$$\boldsymbol{x}_{\mathrm{a}} = \max\left(\frac{Q[K_{\mathrm{init}}, K_{\mathrm{w}}]^T}{\sqrt{d}}, \mathrm{dim} = -1\right) \quad (1)$$

where Q is the query matrix, $[K_{\mathrm{init}}, K_{\mathrm{w}}]$ is the concatenated key vectors, K_{init} corresponds to the initial tokens, K_{w} corresponds to the local window, both selected as blocks for computation. The resulting $\boldsymbol{x}_{\mathrm{a}}$ is the highest score observed within the structurally important regions, which we define as the anchor, as detailed in Algorithm 1.

3.2 Difference-aware Stripe Sparsity Identification

Numerous prior studies (Zhang et al., 2023; Yang et al., 2024; Li et al., 2024) have observed pronounced column-wise correlations in attention score distributions—that is, across multiple consecutive queries, only a small subset of keys consistently receives high attention. However, prior studies (Tang et al., 2024) and our analysis in Section 2.2.1 indicate that such column-wise correlations are not always stable or reliable, often exhibiting vanishing–reappearing behaviors. Inspired by this observation, our strategy shifts focus to global information: for each query segment, we individually identify its corresponding keys and values, rather than relying on a subset of queries to determine a single global key–value set (Jiang et al., 2024; Lai et al., 2025).

As discussed in Section 2.1, to efficiently identify these coordinates, we compress queries through block-average compression and compute their dot product with all keys. The result is directly compared to the average anchor value, $\mathrm{avgpool}(\boldsymbol{x}_{\mathrm{a}})$, from Equation 1 through numerical difference. By setting a hyperparameter θ, we compute only the discrete keys and values whose difference is below this threshold. This approach outperforms static top-k strategies, achieving performance consistent with dynamic top-cdf strategies while avoiding costly sorting operations.

We define the sparsity mask as:

$$\mathrm{mask} = \mathbb{I}\left(\mathrm{avgpool}(\boldsymbol{x}_{\mathrm{a}}) - \frac{\mathrm{avgpool}(Q)K^{\top}}{\sqrt{d}} \leq \theta\right)$$

$$\mathcal{S} = \{(i,j) \mid \mathrm{mask}(i,j) = 1\} \quad (2)$$

where $\boldsymbol{x}_{\mathrm{a}}$ is the approximate highest attention score, $\mathrm{avgpool}(\boldsymbol{x}_{\mathrm{a}})$ is its pooled average, $\mathrm{avgpool}(Q)$ is the pooled queries, θ is the comparison threshold, $\mathrm{mask} \in \{0,1\}^{n \times m}$ is the binary mask, $\mathcal{S}$ is the set of coordinates to be activated, and $\mathbb{I}(\cdot)$ is the indicator function. The detailed implementation is provided in Algorithm 2.

Models	Methods	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
		NarrQA	Qasper	MF-en	HotpotQA	2Wiki	Musique	GovRep	QMSum	MNews	TREC	Trivia	SAMSum	PCount	PR-en	Lcc	RP-P	
LLaMA	Full-attn	31.44	25.07	29.40	16.89	17.00	11.79	34.22	23.25	26.69	72.50	91.65	43.74	5.95	98.20	54.04	51.49	39.58
	StreamingLLM	21.27	23.48	24.05	14.26	13.43	8.46	33.47	22.28	26.76	66.50	90.32	**44.46**	**7.26**	38.24	54.55	52.56	33.83
	Vertical_Slash	20.87	**24.54**	26.19	17.12	14.37	8.38	32.84	22.33	26.85	63.50	**91.38**	44.12	0.98	**98.61**	54.22	36.41	36.48
	FlexPrefill	**28.31**	23.79	28.78	**19.24**	16.22	10.58	33.60	**22.95**	**27.06**	70.50	90.74	43.81	1.37	77.50	54.23	**54.09**	36.66
	Ours	27.79	23.82	**28.86**	16.29	**16.84**	**11.74**	**34.50**	22.94	27.01	**72.50**	90.67	43.82	3.53	96.92	**54.72**	49.65	**38.23**
Qwen	Full-attn	11.53	13.99	31.83	10.88	10.02	7.12	32.52	20.65	22.58	71.50	89.47	46.68	3.92	98.42	59.63	66.57	37.33
	StreamingLLM	11.70	13.68	31.39	11.34	9.77	5.94	32.63	19.85	**22.52**	72.00	89.02	45.76	**4.18**	73.83	59.22	65.28	35.51
	Vertical_Slash	10.70	13.40	31.59	11.30	9.87	**8.06**	**32.70**	20.65	22.47	70.50	**89.73**	**46.00**	3.46	94.25	**60.21**	**66.36**	36.51
	FlexPrefill	8.73	13.91	29.96	**11.36**	8.76	6.69	32.16	**21.08**	22.37	70.50	88.29	45.66	2.03	71.67	58.94	60.68	34.90
	Ours	**14.57**	**14.23**	**32.18**	10.73	**9.93**	7.24	32.21	20.76	22.46	**72.50**	89.05	45.69	3.99	**94.58**	59.28	65.27	**37.17**

Table 2: Accuracy (%) of different attention mechanisms across models on LongBench, where the bold values indicate the optimal configurations of sparse strategies. Our method shows only a marginal accuracy gap compared to full attention, while consistently achieving the best performance among various sparse strategies.

Models	Methods	4k	8k	16k	32k	64k	128k	Avg
LLaMA	Full-attn	95.67	93.75	93.03	87.26	84.37	78.13	88.70
	Streaming LLM	**96.62**	92.06	84.54	66.77	46.69	37.03	70.61
	Vertical_Slash	95.81	92.82	93.26	88.96	**85.09**	58.18	85.69
	FlexPrefill	95.46	93.18	93.53	**90.02**	84.73	**75.03**	**88.66**
	Ours	95.98	**93.27**	**93.67**	87.79	84.53	74.91	88.36
Qwen	Full-attn	94.92	93.01	92.31	86.54	66.76	22.72	76.04
	Streaming LLM	93.74	90.91	74.39	57.81	25.48	15.88	59.70
	Vertical_Slash	94.91	92.16	**92.17**	**85.59**	60.10	24.78	74.95
	FlexPrefill	93.04	90.69	90.16	80.37	40.42	25.43	70.01
	Ours	**94.98**	**92.86**	89.74	84.68	**66.79**	**25.71**	**75.79**

Table 3: Accuracy (%) of different attention mechanisms across models on the RULER benchmark. At the 128k context length, the Qwen model performs poorly, and our method does not achieve the highest end-to-end accuracy across all tests; however, it still demonstrates strong competitiveness in terms of overall accuracy.

3.3 Fine-Grained Sparse Computation

In contrast to prior strategies that load contiguous key-value blocks, our fine-grained sparse computation methodology selectively loads multiple discrete key-value pairs based on discrete key-value coordinates. Throughout the computational process, we adhere to the sharding strategy of FlashAttention, employing the same computation logic. However, compared to block-sparse approaches, our discrete key-value loading, as discussed in Section 2.1.2, achieves a higher sparsity rate due to lower granularity with negligible additional overhead, thereby significantly enhancing the efficiency of sparse computation.

To formalize the fine-grained sparse computation, we construct the reduced key and value sets by discretely loading key-value pairs based on the sparse coordinate set $\mathcal{S}$ from Equation 2. The index set $\mathcal{I}$ is defined as:

$$\mathcal{I} = \{j \mid (i, j) \in \mathcal{S}\}, \tag{3}$$

and the reduced key and value sets are constructed as:

$$\begin{aligned} K' &= \text{load\_discrete}(K, \mathcal{S}) \\ V' &= \text{load\_discrete}(V, \mathcal{S}) \end{aligned} \tag{4}$$

where $\text{load\_discrete}(M, \mathcal{S}) = \{M[j, :] \mid j \in \mathcal{I}\}$ denotes selecting the key or value rows from the matrix M (e.g., K or V) corresponding to the indices in $\mathcal{I}$. The sparse attention output is then computed as:

$$\text{Output} = Attn(Q, K', V') \tag{5}$$

where $Attn(Q, K', V')$ denotes the attention computation, with the granularity of key-value loading modified from contiguous blocks (as in FlashAttention(Dao et al., 2022)) to discrete keys and values based on the coordinates in $\mathcal{S}$. The detailed implementation is provided in Algorithm 3.

3.4 Kernel Optimization and Algorithm

To further accelerate sparse attention computation, we implement kernel-level optimizations across all algorithms with two primary objectives: (1) maximizing parallel computation capacity and (2) avoiding additional memory overhead. To this end, we introduce an extra hyperparameter, *step*, which enables simultaneous identification of coordinates corresponding to *step* query blocks. If any of these blocks contain a key that satisfies the condition defined in Equation 2, all *step* consecutive blocks are marked as active for computation,

thereby enabling unified processing and improved parallelism. Meanwhile, to reduce redundant computation, we temporarily cache the intermediate results generated in Section 3.1 and reuse them in Section 3.3. This design maximizes computational efficiency while incurring only negligible memory overhead compared to the original key–value cache. The complete implementation is provided in Algorithms 1, 2 and 3.

4 Experiment

4.1 Setup

Models Our evaluation is conducted on two advanced large language models (LLMs) that natively support up to 128K context length in their pretrained form: (i) **LLaMA-3.1-8B-Instruct** (**?**), (ii) **Qwen2.5-7B-Instruct** (Qwen et al., 2025).

Benchmark We evaluate models on three representative long-context benchmarks, each designed to test different aspects of long-context understanding and retrieval: (i) **LongBench** (Bai et al., 2024), a multilingual, multi-task benchmark covering question answering, summarization, classification, and retrieval, with diverse input formats; (ii) **RULER** (Hsieh et al., 2024), a synthetic benchmark that enables controlled variations in context length and reasoning complexity, including tasks such as multi-hop tracing and aggregation; (iii) **Needle-in-a-Haystack** (Kamradt, 2023), a stress test designed to evaluate accurate retrieval performance in ultra-long contexts.

Baseline We evaluate four baselines for accelerating prefill attention: (i) **Full-attn**, dense attention implemented via FlashAttention (Dao et al., 2022); (ii) **Vertical_Slash** (Jiang et al., 2024), which selects a fixed set of important vertical and slash positions; (iii) **StreamingLLM** (Xiao et al., 2024), retaining only key tokens from initial and local window regions; (iv) **FlexPrefill** (Lai et al., 2025), a dynamic method selecting attention blocks based on top-cdf scoring, representing recent state-of-the-art.

Implementation All experiments are conducted on a single NVIDIA A100 GPU with 80GB memory, leveraging Triton (Tillet et al., 2019) for optimized GPU computations. To ensure fair comparison, all methods adopt a uniform block size of 128. Across all datasets, our method and FlexPrefill use consistent hyperparameter settings: for ours, we set $\theta = 12$ and step $= 16$; for FlexPrefill, we use $\gamma = 0.95$, $\tau = 0.1$, and min_budget $= 1024$. In our parameter configuration, we choose $\theta = 12$, under which the sparsified positions have almost no impact on the final output. For LongBench, which has relatively shorter average sequence lengths, StreamingLLM uses a global window and a local window of 1024, and Vertical_Slash sets both vertical and slash window sizes to 1024. For other datasets, StreamingLLM adopts a global window of 1024 and a local window of 8192, while Vertical_Slash uses a vertical window of 1024 and a slash window of 8192. In the latency-recall evaluation, we uniformly choose to generate data using the ruler and report the averaged results.

4.2 Result

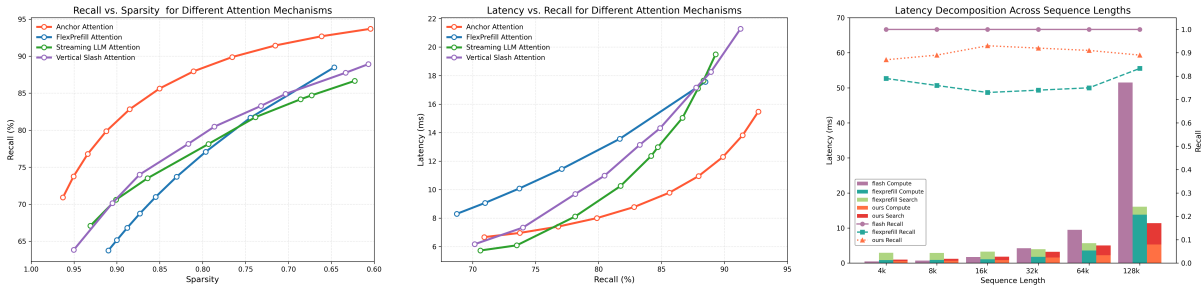
Longbench To demonstrate the applicability of our method to nearly all input scenarios, we selected the LongBench benchmark for accuracy evaluation. LongBench encompasses a variety of tasks that exhibit input diversity, testing whether our method maintains high accuracy across different inputs. The accuracy results are presented in Table 2.

Ruler To demonstrate the potential of our approach for large language models handling varying context lengths, we conducted evaluations on multiple methods using the ruler benchmark. Table 3 shows that, as context length increases, our method consistently maintains accuracy close to that of full KV computations.

Needle-in-a-Haystack As shown in Figure 7, we present the results of the Needle-in-a-Haystack task across different context lengths and depth percentages. The results indicate that both our method and FlexPrefill can dynamically adapt the sparsity rate based on input variations, achieving performance comparable to full attention. In contrast, the static strategy Vertical_Slash shows a noticeable accuracy drop as the context length increases.

Recall vs. Sparsity We adjust the hyperparameters of different methods to obtain varying sparsity rates and compare the recall performance of different strategies under each sparsity level. As shown in Figure 6a, our method achieves the highest sparsity rate under the same recall level.

Latency vs. Recall Prior work primarily differs in search strategies, with distinctions arising from the blocks requiring computation. Our method abandons block-level sparsity strategies, instead adopting a finer-grained computation strategy that loads multiple discrete keys and values at once. As illustrated in Figure 6b, at the same recall level, our



(a) Recall vs. Sparsity (b) Latency vs. Recall (c) Latency vs. Length

Figure 6: Performance metrics for recall, sparsity, and efficiency across different methods. Figures (a), (b), and (c) are generated from a random sample of the Ruler benchmark to illustrate the effectiveness of different approaches. Figures (a) and (b) correspond to the 128k length. For sparsity and latency metrics, we report the average computation time per head. The results show that our method significantly outperforms other sparse strategies in terms of recall.

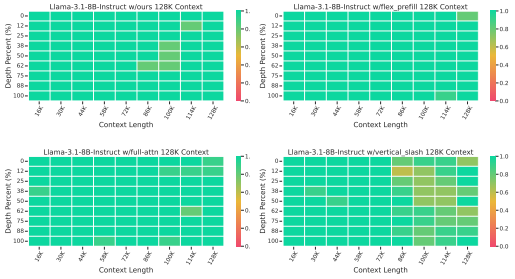


Figure 7: Comparison of attention patterns on Needle-in-a-Haystack tasks (128K context), where our method achieves results comparable to full attention and demonstrates strong performance.

Anchor Attention	θ	Sparsity (%)	Recall (%)	Time (ms)
With Anchor	10.0	97%	70.9	5.7
	11.0	93%	76.8	6.4
	12.0	89%	82.8	8.2
	13.0	81%	88.0	10.9
	14.0	72%	91.4	13.8
	15.0	61%	94.7	19.3
Without Anchor	10.0	83%	69.5	9.3
	11.0	69%	83.7	14.6
	12.0	52%	90.2	29.5
	13.0	47%	95.8	41.3
	14.0	18%	96.2	49.7
	15.0	3%	98.5	57.2

Table 4: Ablation study of Anchor Attention. Results are averaged over all heads using a 128k context length. The results demonstrate that the selection of anchors is effective, as it allows dynamically setting the maximum value to compare for each query. In contrast, static linear schemes do not achieve better results, indicating the effectiveness of dynamic anchor selection.

strategy significantly outperforms other methods in terms of time efficiency.

Latency vs. Length Compared to prior strategies, our approach considers the entire region during search. This higher search overhead also brings us more accurate recognition, which is reflected in the recall curves and the computation time section. As shown in Figure 6c, our method incurs additional recognition time in most cases, but it achieves a higher important recognition ratio, thereby optimizing overall time efficiency and recall.

4.3 Ablation Study

Anchor Importance In this section, we assess the impact of introducing anchors when searching for important tokens by comparing sparsity, recall, and computation time under different values of θ. As shown in Table 4. The original attention(With Anchor) consistently achieves high recall rates while maintaining impressively low sparsity, indicating effective attention guidance. In contrast, the Without Anchor configuration, which sets the anchor as a zero tensor in implementation, requires significantly higher sparsity to reach comparable recall levels. This suggests that fixed thresholding alone, without anchor guidance, is less adept at capturing the global attention distribution efficiently, resulting in a less optimal sparsity-recall balance.

5 Related Work

LLM Inference Acceleration Inference acceleration techniques aim to reduce the latency and memory overhead of large language models (LLMs) during text generation. At the system level, FlashAttention (Dao et al., 2022) significantly improves attention computation efficiency by optimizing memory access patterns, while RingAttention (Liu et al., 2023) distributes attention workloads across multi-

ple devices to achieve parallel acceleration. PagedAttention (Kwon et al., 2023) further enhances overall inference performance through efficient KV cache management. In terms of model compression and storage optimization, quantization techniques are widely employed to reduce model size and memory bandwidth requirements, thereby accelerating inference. Specifically for KV cache compression, research has progressed from early per-token quantization methods such as Atom (Zhao et al., 2024), to channel-wise quantization like KIVI (Liu et al., 2024b), and more recently to non-uniform quantization schemes such as KVQuant (Hooper et al., 2024).

Sparse Attention The quadratic complexity of attention has driven extensive research into sparse attention strategies to improve the inference efficiency of large language models (LLMs). Importantly, attention distributions in LLMs are inherently sparse—many attention weights are close to zero and can be safely pruned without significantly affecting model performance (Child et al., 2019). More recent methods, such as H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024) prune unimportant tokens by comparing cumulative attention scores. Although partially effective, these methods offer limited acceleration benefits during the prefill stage. StreamingLLM (Xiao et al., 2024) significantly improves efficiency by retaining only initial and recent tokens, but often misses critical information from intermediate regions. MInference (Jiang et al., 2024) accelerates the prefill stage by applying statically determined attention patterns, but such static designs are often suboptimal for diverse and dynamic inputs. FlexPrefill (Lai et al., 2025) improves adaptivity via runtime-driven dynamic pattern selection, yet relies heavily on local information, limiting its ability to capture globally important positions. Recently, research has shifted toward building general-purpose sparse attention frameworks rather than designing architectures tailored specifically to LLM characteristics. For example, SpargeAttn(Zhang et al., 2025) leverages similarity-based filtering and quantization to accelerate attention, while X-Attention(Xu et al., 2025) introduces an antidiagonal scoring mechanism to efficiently prune irrelevant blocks. Furthermore, most existing methods rely on block-level granularity, where block size fundamentally constrains the achievable sparsity ceiling. Therefore, there is an urgent need for a lower-granularity sparse attention mechanism with a stronger emphasis on global context, in order to mitigate the increasingly heavy computational burden during the prefill stage as context lengths continue to grow.

6 Conclusion

In this work, we propose **AnchorAttention**, a difference-aware, dynamic sparse attention mechanism designed to address the computational challenges faced by Large Language Models (LLMs) during the prefill phase under long-context settings. The method efficiently identifies critical attention regions at a finer *stripe-level* granularity.

To further improve speed, we implement all operators at the kernel level. By combining pattern-based anchor computation, difference-aware stripe sparsity identification, and fine-grained sparse computation, **AnchorAttention** achieves higher sparsity and superior computational efficiency compared to existing methods. At a sequence length of 128k, it achieves a 1.44× speedup while maintaining a higher recall rate.

7 ACKNOWLEDGMENTS

Sponsored by Tencent Basic Platform Technology Rhino-Bird Focused Research Program.

Limitations

Our evaluation is limited to the LLaMA-3.1-8B-instruct and Qwen2.5-7B-instruct models, and we have not yet validated the generality of AnchorAttention across a broader range of architectures and model scales; future work will extend our analysis to additional models. Additionally, we do not account for the importance of slash and row-wise patterns, as our design prioritizes maximizing parallelism while ensuring high recall rates. Furthermore, this work focuses exclusively on the prefill phase of attention computation and does not analyze the impact or adaptivity of our method during the decode phase; subsequent studies will investigate performance and sparsity behavior during generation.

Ethics Statement

We believe this work raises no ethical concerns. Attention is a key component in Transformers, widely used in Large Language Models (LLMs). Therefore, accelerating the execution of attention is beneficial for developing LLM applications that address diverse societal challenges.

References

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.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. 2024. Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and Yuxiong He. 2024. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization.

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

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

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models.

Greg Kamradt. 2023. Llmtest needle in a haystack - pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`. Accessed: [Insert Date].

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

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

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. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. 2024a. Retrievalattention: Accelerating long-context llm inference via vector retrieval.

Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring attention with blockwise transformers for near-infinite context.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.

Maxim Milakov and Natalia Gimelshein. 2018. Online normalizer calculation for softmax.

Libo Qin, Qiguang Chen, Xiachong Feng, Yang Wu, Yongheng Zhang, Yinghui Li, Min Li, Wanxiang Che, and Philip S. Yu. 2024. Large language models meet nlp: A survey.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report.

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

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, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. 2025. Xattention: Block sparse attention with antidiagonal scoring.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference.

Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. 2025. Lserve: Efficient long-sequence llm serving with unified sparse attention.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. 2025. Spargeattn: Accurate sparse attention accelerating any model inference.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate llm serving.

Shuang Zhou, Zidu Xu, Mian Zhang, Chunpu Xu, Yawen Guo, Zaifu Zhan, Sirui Ding, Jiashuo Wang, Kaishuai Xu, Yi Fang, Liqiao Xia, Jeremy Yeung, Daochen Zha, Genevieve B. Melton, Mingquan Lin, and Rui Zhang. 2024. Large language models for disease diagnosis: A scoping review.

A Sparsity Heatmap Comparison

Figure 4 presents the per-layer, per-head recall distributions on the LLaMA-3.1-8B-instruct model using the 128k ruler datasets. In Figure 8, we further visualize the sparsity levels achieved under this target recall for different identification strategies. The results indicate that our proposed Difference-Aware strategy achieves sparsity patterns comparable to those of top-cdf while maintaining similar recall performance.

B Dynamic Sparsity Heatmap

To demonstrate the dynamic nature of the heatmap, we selected a distinct dataset with the same length of 128k. The recall rates under different sparsity strategies are shown in Figure 9, with the corresponding sparsity rates depicted in Figure 10. It is evident that, as the input changes, both the top-cdf and difference-aware methods can effectively capture variations in sparsity rates.

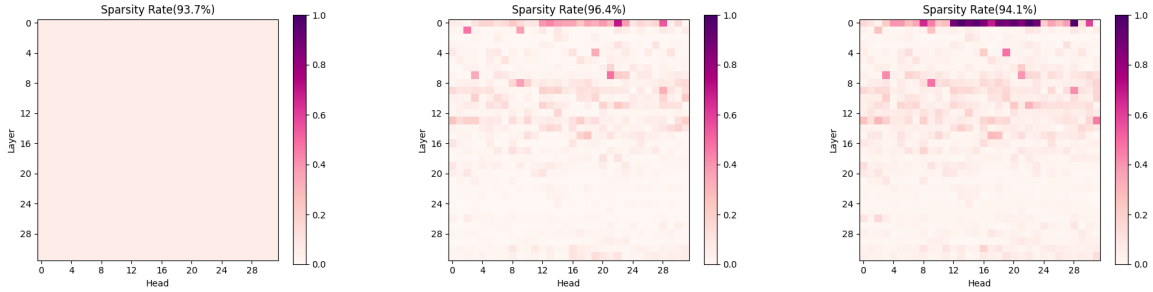
C Algorithm

We provide the complete pseudocode of our proposed sparse attention inference pipeline, consisting of three key stages:

Algorithm 1: Anchor Computation. This algorithm performs efficient block-wise attention to obtain an approximate estimation of the attention result, which is used later for sparsity identification. The query matrix Q is divided into blocks Q_i and interacts only with a small number of key-value blocks (e.g., the initial block and a local window). The accumulated attention values Acc_i, normalization terms L_i, and maximum logits M_i are computed and cached. These intermediate results are reused in the final sparse attention step to avoid redundant computation.

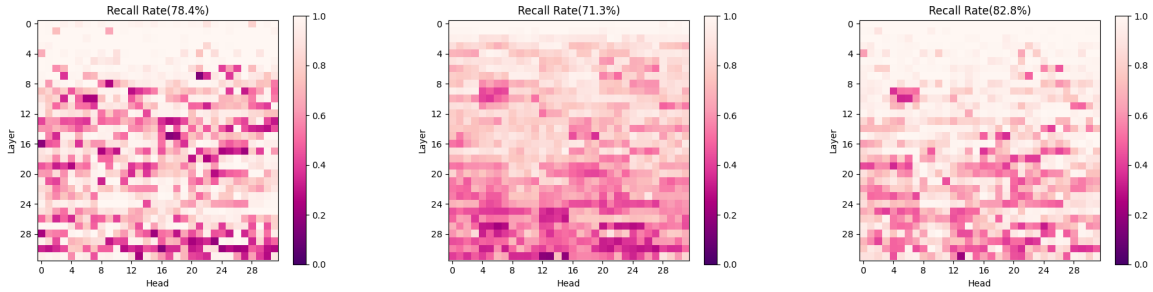
Algorithm 2: Stripe Sparsity Identification. Based on the averaged queries and approximated attention output from the previous step, this algorithm identifies informative positions through a lightweight thresholding mechanism. By comparing the approximated anchor score x_a with new attention estimates, it selects positions with scores close to the anchor. This enables the construction of stripe-wise sparse indices F_idx without computing full attention maps, greatly improving efficiency.

Algorithm 3: Sparse Attention Computation. This stage computes the final attention output using only the key/value blocks selected via sparse indexing. For each query block Q_i, the algorithm loads its corresponding anchor values (M_i, L_i, Acc_i) and incrementally accumulates the attention using the sparse key-value entries. This computation avoids redundant processing and yields high sparsity while maintaining high recall and accuracy.



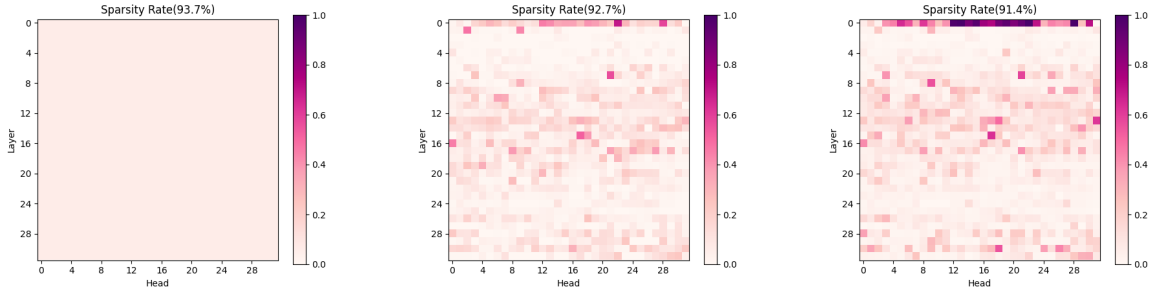
(a) Top-K (4096) (b) Top-CDF (0.95) (c) Difference-Aware (11)

Figure 8: Sparsity heatmaps under different sparsity strategies. The recall heatmap corresponds to Figure 4.



(a) Top-K (4096) (b) Top-CDF (0.95) (c) Difference-Aware (11)

Figure 9: Recall heatmaps under different sparsity identification strategies.



(a) Top-K (4096) (b) Top-CDF (0.95) (c) Difference-Aware (11)

Figure 10: Sparsity heatmaps for different sparsity strategies.

Algorithm 1 Anchor Computation

Require: $Q, K, V \in \mathbb{R}^{N\times d}$ (FP16), block sizes b_q, b_{kv}, step size $step$
Divide Q into $T_m = N/b_q$ blocks $\{Q_i\}$; K, V into $T_n = N/b_{kv}$ blocks $\{K_j\}, \{V_j\}$
for $i = 1$ to T_m **do**
  Load Q_i, K_1, V_1 into shared memory
  Compute initial attention: $qk \leftarrow Q_i \cdot K_1$
  $m \leftarrow \max(qk, \text{dim} = -1)$
  $p \leftarrow \exp(qk - m)$, $l \leftarrow \sum(p, \text{dim} = -1)$, $acc \leftarrow p \cdot V_1$
  Determine local window range:
    $j_{\text{start}} \leftarrow \max(2, \lfloor (i-1)/\text{step} \rfloor \cdot \text{step} \cdot (b_q/b_{kv}))$
    $j_{\text{end}} \leftarrow i \cdot (b_q/b_{kv})$
  for $j = j_{\text{start}}$ to j_{end} **do**
    Load K_j, V_j into shared memory
    Compute $qk \leftarrow Q_i \cdot K_j$, $m' \leftarrow \max(m, \max(qk))$
    $p \leftarrow \exp(qk - m')$, $\alpha \leftarrow \exp(m - m')$
    $l \leftarrow l \cdot \alpha + \sum(p)$, $acc \leftarrow acc \cdot \alpha + p \cdot V_j$
    Update $m \leftarrow m'$
  end for
  Write $M_i \leftarrow m$, $L_i \leftarrow l$, $Acc_i \leftarrow acc$
end for
return M, L, Acc

Algorithm 2 Stripe Sparsity Identification

Require: $Q, K \in \mathbb{R}^{N\times d}$ (FP16), anchor score Acc, block sizes b_q, b_{kv}, threshold θ, step size $step$
Compute averaged query $Q_{\text{mean}} \leftarrow \text{avgpool}(Q, b_q)$
Compute anchor average $x_a \leftarrow \text{avgpool}(Acc, b_q)$
Divide Q_{mean} into $T_m = N/(b_q \cdot \text{step})$ blocks $\{Q_i^m\}$
Divide K into $T_n = N/b_{kv}$ blocks $\{K_j\}$
for $i = 1$ to T_m **do**
  Initialize $f_c \leftarrow 0$, $f_{\text{idx}} \leftarrow \emptyset$
  $j_{\text{end}} \leftarrow (i-1) \cdot \text{step} \cdot (b_q/b_{kv})$
  for $j = 2$ to j_{end} **do**
    Load K_j
    Compute $qk \leftarrow Q_i^m \cdot K_j$
    mask $\leftarrow (x_a - qk) < \theta$
    Append matching indices to f_{idx}, count to f_c
  end for
  Write $F_{\text{idx}}^{(i)} \leftarrow f_{\text{idx}}$, $\quad F_c^{(i)} \leftarrow f_c$
end for
return F_{idx}, F_c

Algorithm 3 Sparse Attention Computation (Reusing Anchor and Stripe Outputs)

Require: Query Q, Key K, Value $V \in \mathbb{R}^{N\times d}$ (FP16), precomputed M, L, Acc (from Alg. 1), and sparse indices F_{idx}, F_c (from Alg. 2); block sizes b_q, b_{kv}; step size $step$
Divide Q into $T_m = N/b_q$ blocks $\{Q_i\}$
Divide M, L, Acc into $\{M_i\}$, $\{L_i\}$, $\{Acc_i\}$
Divide F_c, F_{idx} into $\{F_c^{(k)}\}$, $\{F_{\text{idx}}^{(k)}\}$ where $k = \lfloor (i-1)/\text{step} \rfloor$
for $i = 1$ to T_m **do**
  Load Q_i, and corresponding M_i, L_i, Acc_i
  Initialize $m \leftarrow M_i$, $l \leftarrow L_i$, $acc \leftarrow Acc_i$
  Let $k \leftarrow \lfloor (i-1)/\text{step} \rfloor$
  # Simultaneously load multiple discrete coordinate chunks from $F_{idx}^{(k)}$
  for each index chunk f_{idx}^j in $F_{\text{idx}}^{(k)}$ **do**
    Load sparse key/value: $K_j = K[f_{\text{idx}}^j]$, $V_j = V[f_{\text{idx}}^j]$
    Compute $qk = Q_i \cdot K_j$, $m' = \max(m, \max(qk))$
    $p = \exp(qk - m')$, $\alpha = \exp(m - m')$
    $l = l \cdot \alpha + \sum(p)$, $acc = acc \cdot \alpha + p \cdot V_j$
    Update $m = m'$
  end for
  Write output $O_i = acc/l$
end for
return Final attention output O