MInference: Accelerating Pre-filling for Long-Context LLMs via Dynamic Sparse Attention

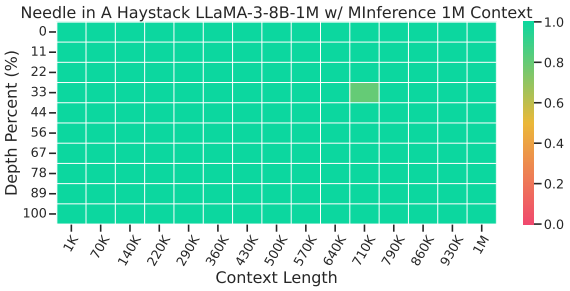
Huiqiang Jiang [*1] **Yucheng Li** [*2] **Chengruidong Zhang** [*1] **Qianhui Wu** [1] **Xufang Luo** [1] **Surin Ahn** [1]
Zhenhua Han [1] **Amir H. Abdi** [1] **Dongsheng Li** [1] **Chin-Yew Lin** [1] **Yuqing Yang** [1] **Lili Qiu** [1]

Abstract

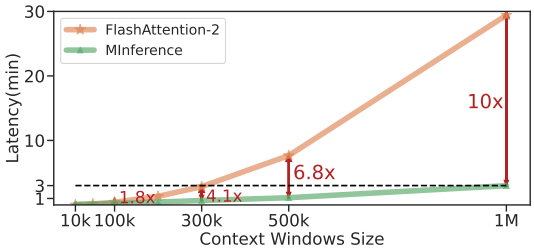
The computational challenges of LLM inference remain a significant barrier to their widespread deployment, especially as prompt lengths continue to increase. Due to the quadratic complexity of the attention computation, it takes 30 mins for an 8B LLM to process a prompt of 1M tokens (i.e., the pre-filling stage) on a single A100 GPU. Existing methods for speeding up pre-filling often fail to maintain acceptable accuracy or efficiency when applied to long-context LLMs. To address this gap, we introduce **MInference** (*Million-tokens Inference*), a sparse calculation method designed to accelerate pre-filling of long-sequence processing. Specifically, we identify three unique patterns in long-context attention matrices—the *A-shape*, *Vertical-Slash*, and *Block-Sparse*—that can be leveraged for efficient sparse computation on GPUs. We determine the optimal pattern for each attention head offline and dynamically build sparse indices based on the assigned pattern during inference. With the pattern and sparse indices, we perform efficient sparse attention calculations via our optimized GPU kernels to significantly reduce the latency in the pre-filling stage of long-context LLMs. Our proposed technique can be directly applied to existing LLMs without any modifications to the pre-training setup or additional fine-tuning. By evaluating a wide range of downstream tasks, including InfiniteBench, RULER, PG-19, and Needle In A Haystack, and models including LLaMA-3-1M, GLM-4-1M, Yi-200K, Phi-3-128K, and Qwen2-128K, we demonstrate that MInference effectively reduces inference latency by up to 10× for pre-filling on an A100, while maintaining accuracy. Our code is available at `https://aka.ms/MInference`.

[*]Equal contribution [1]Microsoft Corporation [2]University of Surrey. Correspondence to: Huiqiang Jiang <hjiang@microsoft.com>.

Work presented at the ES-FoMo Workshop at ICML 2024, Vienna, Austria. Copyright 2024 by the author(s).



(a) Needle In A Haystack



(b) Latency Speedup

Figure 1. Attention weights, especially in long-context, exhibit up to 96.8% sparsity in 128K. We propose **MInference**, which uses dynamic sparse attention to accelerate the pre-filling stage of long-context LLM inference. It achieves up to 10x speedup for 1M contexts on an A100, as shown in (b), and maintains or outperforms baselines, as demonstrated by the Needle In A Haystack (Kamradt, 2023) in (a) on LLaMA-3-8B-1M (Gradient, 2024).

1. Introduction

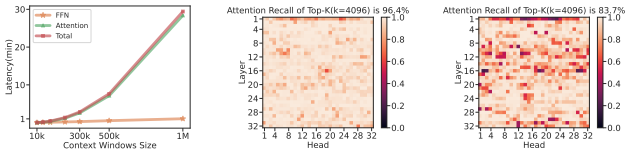
Large language models (LLMs) have entered the era of long-context processing, with some of them supporting context windows ranging from 128K to 10M tokens (Gradient, 2024; Reid et al., 2024; Liu et al., 2024b; Young et al., 2024; Abdin et al., 2024; DeepSeek-AI, 2024). These extended context windows enable LLMs to unlock a multitude of complex real-world applications, such as repository-level code understanding (Bairi et al., 2023; Jimenez et al., 2023; Park et al., 2023), long-document question-answering (Caciularu et al., 2023; Li et al., 2024a), extreme-label in-context learning (Li et al., 2024a), and long-horizon agent tasks (Weng, 2023).

However, due to the quadratic complexity of attention, it can take several minutes for the model to process the input prompt (i.e., the pre-filling stage) and then start to produce the first token, which leads to unacceptable Time To First Token experience, thus greatly hinders the wide application of long-context LLMs. As shown in Fig. 2a, when serving LLaMA-3-8B on a single A100 machine, the model would keep users waiting for 6 minutes to finish the pre-filling stage given a prompt of 300K tokens, and this number increases to 30 minutes for a prompt of 1M tokens. The overhead of self-attention computation exceeds 90% of the total pre-filling latency, which makes it the major bottleneck in long-context processing of LLMs. Previous research has shown that the attention matrices are highly sparse (Liu et al., 2022; Deng et al., 2024), which has led to the development of fixed sparse attention methods such as Longformer (Beltagy et al., 2020) and BigBird (Zaheer et al., 2020). However, prior studies have also noted that attention distributions vary significantly across different inputs (Liu et al., 2022; Likhosherstov et al., 2021). This dynamic nature prevents prior sparse methods from being used directly on long-context LLMs without expensive training or fine-tuning. But if the dynamic sparse attention patterns could be efficiently predicted online, the pre-filling latency of long-context LLMs could be significantly reduced by calculating only the most important part of the attention weights.

Building upon this idea, we present **MInference**, a technique that reduces 95% of FLOPs in the attention computation to significantly accelerate the pre-filling stage of long-context LLM inference via dynamic sparse attention. Unlike existing dynamic sparse attention methods that introduce large computational overhead to estimate attention patterns with low-rank hidden dimensions (Liu et al., 2022; Ribar et al., 2024), our method is designed specifically for long-context scenarios with minimal overhead in estimation. Specifically, we conduct extensive analysis and identify three general patterns of sparse attention in long-context LLMs: *A-shape* pattern, *Vertical-Slash* pattern, and *Block-Sparse* pattern. Based on these findings, we introduce a kernel-aware search method to assign the optimal attention pattern for each head. Importantly, instead of fixed attention masks in prior studies, we perform an efficient online approximation to build a dynamic sparse mask for each head according to their assigned pattern and particular inputs. For example, to build a dynamic sparse mask for a specific prompt on one *Vertical-Slash* head, we use a partial of attention weight consisting of the last last_q query and key vectors to estimate the most important indices of the vertical and slash lines globally on the attention matrix. For *Block-Sparse* heads, we perform mean pooling on both query and key vectors in blocks of 64 and calculate the block-level attention weights to determine the most important blocks and thereby obtain a block-sparse dynamic mask. After obtaining the dynamic sparse mask, three optimized GPU kernels are used, which we developed for the above three sparse patterns. These kernels are based on the dynamic sparse compilers PIT (Zheng et al., 2023), Triton (Tillet et al., 2019) and FlashAttention (Dao, 2024), which enable extremely efficient computation of dynamic sparse attention.

Extensive experiments are conducted on various Long-context LLMs, including LLaMA-3-1M (Gradient, 2024), GLM-4-1M (GLM et al., 2024), and Yi-200K (Young et al., 2024), across benchmarks with context lengths over 1M tokens, such as InfiniteBench (Zhang et al., 2024a), RULER (Hsieh et al., 2024), Needle In A Haystack (Kamradt, 2023), and PG-19 (Rae et al., 2020). Needle In A Haystack was also tested on Phi-3-Mini-128K (Abdin et al., 2024) and Qwen-2-128K (Yang et al., 2024). Results show that MInference speeds up the pre-filling stage by up to $10\times$ for 1M contexts with LLaMA-3-8B on a single A100, reducing latency from 30 minutes to 3 minutes per prompt, while maintaining or improving accuracy.

2. Attention Heads: Dynamic, Sparse, and Characteristic



(a) Attention incurs heavy cost. (b) Attention is sparse. (c) Sparsity of attention is dynamic.

Figure 2. (a) Latency breakdown of the pre-filling stage. (b) How much attention scores can top-k (k=4096) columns cover in a 128k context. (c) Less attention scores are retrieved when reusing the top-k indices from another examples, indicating its dynamic nature. Visualizations are based on LLaMa-3-8B with a single A100.

2.1. Attention is Dynamically Sparse

The sparsity of attention weights in pre-trained LLMs, especially in long-context scenarios, has been well-documented (Liu et al., 2022; Ribar et al., 2024; Liu et al., 2023b; Xiao et al., 2024b). As shown in Fig. 2b, for an attention matrix of size $128k \times 128k$, retaining only the top 4k columns recalls 96.8% of the total attention. In other words, each token is attending to a limit number of tokens despite the long sequence it is processing.

On the other hand, although the sparse nature of attention matrices is shared across different inputs, the exact distributions of sparse pattern are highly dynamic. That is to say, although a token in a specific position only attends to a limited number of tokens in self-attention, the exact tokens attended to vary significantly across different prompts. This

dynamism has been mathematically demonstrated in prior studies (Likhosherstov et al., 2021; 2023). As depicted in Fig. 2c, if we take the top 4k columns found in Fig. 2b and apply it on another prompt of 128k length, the recall of attention would drop largely to 83.7%.

2.2. Attention Sparsity Exhibits Patterns

Table 1. Comparison of different sparse patterns.

Patterns	A-shape	Vertical-Slash	Block-Sparse	Top-K
Spatial Distribution	Static structured	Dynamic structured	Dynamic structured	Dynamic fine-grained
Latency on GPU	Low	Medium	Low	High
Time to build the index	Zero	Small	Small	High

Although the sparsity distribution of attention matrix is dynamic, previous works (Xiao et al., 2024b; Han et al., 2024) have shown that they exhibit certain patterns in the two-dimensional space such as spatial clustering. Through our analysis of long-context prompts of various lengths and tasks, we have categorized such attention sparse patterns into the *A-shape*, *Vertical-Slash* (VS), and *Block-Sparse* patterns, as shown in Fig. 4a and Fig. 3. Table 1 details the characteristics and differences between these three patterns.

***A-shape* pattern** The attention weights of these types of heads are concentrated on initial tokens and local windows (Xiao et al., 2024b; Han et al., 2024), exhibiting relatively higher stability.

***Vertical-Slash* (VS) pattern** The attention weights are concentrated on specific tokens (vertical lines) and tokens at fixed intervals (slash lines). The positions of vertical and slash lines dynamically change with the context content and exhibit a certain sparsity, making them difficult to be encompassed by local windows and *A-shape* patterns.

***Block-Sparse* pattern** This sparsity pattern is the most dynamic, exhibiting a more dispersed distribution. Despite its dynamism, the attention weights are spatial clustered, which we identify as the block-sparse pattern. We analyzed the distances between non-zero attention weights and their top-k nearest non-zero neighbors within a 128k prompt as shown in Fig. 4b. The results indicate that across layers and heads, the distances between nearest non-zero values are generally concentrated around 5, suggesting a strong spatial clustering of the attention weights.

The point of these three patterns is that we can leverage them to perform highly efficient sparse computing for the attention matrix in long-context LLMs. In Fig. 4c, we test how efficient is our indentified patterns retrieving attention scores with limit computing cost on GPU (FLOPs). First, attention heads are labeled with one of the sparse pattern (detail see §3.1). Then we demonstrate our patterns are significantly more efficient compared to other sparse methods (Ribar et al., 2024; Xiao et al., 2024b; Pagliardini et al., 2024). Specifically, with the same amount of FLOPs, our patterns achieve a notable higher recall on attention scores, which can potentially lead to better accuracy. For example, on attention heads with *Block-Sparse* pattern, previous Top-K methods (Ribar et al., 2024; Xiao et al., 2024b; Pagliardini et al., 2024) fail to capture attention scores efficiently as they are focusing on specific (Top-K) tokens globally, but our pattern is promising in retrieve attention score efficiently and accurately. We example how we use these patterns on long-context LLMs and how we implement optimized GPU kernels for these patterns in §3.

3. MInference

Following the analysis in §2, we propose **MInference** to accelerate the pre-filling stage of long-context LLMs, including: 1) Offline attention pattern identification for each head; 2) Dynamic build of sparse indices w.r.t. the pattern; 3) Sparse attention calculation with optimized GPU kernels.

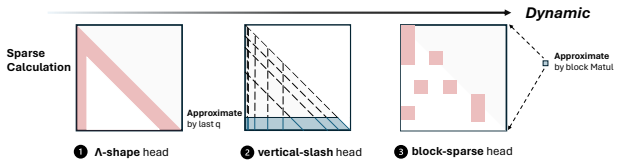


Figure 3. The three sparse methods in MInference.

3.1. Speedup of Long-context LLM Inference via Dynamic Sparse Attention

Kernel-Aware Optimal Sparse Pattern Search To achieve the best accuracy with limited FLOPs budget, we propose an offline Kernel-Aware Optimal Sparse Pattern Search method. In this step, we determine which sparse pattern will be used for each head, and the optimal setting for the pattern in real calculation (e.g., the number of vertical/slash lines in *VS* pattern; or the number of top-k blocks in *BS* patterns). As shown in Algorithm 1, we first create the search space based on a target FLOPs for each pattern, ensuring all potential candidates (i.e., different patterns with different settings) have similar computational cost. *Kernel-aware* here indicates the computational cost reflects the real FLOPs in GPU kernels, instead of conceptual estimations, which is crucial to achieve the optimal acceleration.

Next, we go through the search space with a reference example to decide the optimal pattern and setting. Specifically, we use recall of the attention output as the objective criterion when searching for the best pattern. This approach leverages FlashAttention (Dao, 2024) to reduce GPU memory overhead and incorporates the information from the $\boldsymbol{V}$ matrix, enabling end-to-end selection of the best pattern, which further enhances performance.

Sparsity Indices Approximation and Dynamic Sparse Attention Calculation During the inference stage, we

will perform an online estimation on the attention matrix to dynamically determine the spatial distribution our sparse indices, based on the assigned patterns and the exact input. After that, we conduct the sparse attention computations with our optimized GPU kernels. The implementation details of our kernels can be found in Appendix C.5. Noted that the sparse mask is static for *A-shape* heads, so there is no overhead in building the dynamic masks, and only sparse calculation is required.

(i) Vertical-Slash head. As shown in Algorithm 2, due to the continuity of vertical and slash lines, we matmul the last query vector $\boldsymbol{Q}_{[-\text{last\_q}:]}$ and key vector $\boldsymbol{K}$ to produce the estimated attention matrix $\widehat{\boldsymbol{A}}$, which, in turn, is used to determine the indices for the vertical $\boldsymbol{i}_v$ and slash $\boldsymbol{i}_s$ lines. After obtaining the sparse indices for the vertical and slash lines, we convert them into a sparse format $\boldsymbol{i}_{vs}$. Using these sparse indices, we perform block-sparse calculations of the attention weights and attention output.

(ii) Block-Sparse head. Per Algorithm 3, mean pooling is applied on $\boldsymbol{Q}$ and $\boldsymbol{K}$ to obtain $\widehat{\boldsymbol{Q}}$ and $\widehat{\boldsymbol{K}}$, respectively. The two matrices are multiplied to get the estimated block-level attention weights $\widehat{\boldsymbol{A}}$. Since the mean pooling and matrix multiplication operations are commutative, the resulting attention weights are approximately equivalent to the actual attention weights after mean pooling. This allows us to approximate the actual block-sparse pattern with minimal overhead. Similarly, we build a sparse index $\boldsymbol{i}_b$ and use it to compute the sparse attention weights and attention output.

4. Experiments

In this section, we investigate two questions: **(i) How effective is MInference?** We evaluate our method on three general long-context benchmarks: InfiniteBench (Zhang et al., 2024a), RULER (Hsieh et al., 2024), and the Needle In A Haystack (Kamradt, 2023), as well as the long-context language modeling task (Rae et al., 2020). These benchmarks cover long-context QA, multi-hop QA, math reasoning, aggregation tasks, summarization, retrieval tasks, and code debugging, allowing us to assess MInference's effectiveness across a wide range of long-context scenarios. **(ii) How efficient is MInference?** We delve into the end-to-end latency and its breakdown to evaluate the efficiency of MInference. Additional experimental, latency results, and analysis can be found in Appendix D, F, and G.

InfiniteBench As shown in Table 2, MInference achieves the best overall performance compared to baselines on InfiniteBench. Remarkably, MInference matches or even slightly surpasses the performance of the original full attention baseline on some tasks, despite the significant acceleration it provided. From the perspective of different tasks, our method not only performs well in natural language tasks such as summarization, QA, and code, but also maintains the original model's performance on retrieval-related tasks. Baseline methods such as StreamingLLM, on the contrary, struggle with these retrieval tasks. Additionally, on tasks such as dialogue QA, using local attention mechanisms can better handle these tasks, while our performance is closer to the original results, indicating that our method is not solely based on local windows. Extending the local windows' intervals in StreamingLLM, i.e., w/ dilated and w/ strided, hardly affects the model's performance.

Table 2. Performance of different methods with different base models on InfiniteBench (Zhang et al., 2024a).

Methods	En.Sum	En.QA	En.MC	En.Dia	Zh.QA	Code.Debug	Math.Find	Retr.PassKey	Retr.Number	Retr.KV	Avg.
LLaMA-3-8B-262K	20.2	12.4	67.3	6.0	12.9	22.1	26.6	100.0	100.0	14.4	38.2
StreamingLLM	21.0	8.2	40.2	10.0	10.4	**25.9**	30.0	86.8	5.1	0.8	23.8
StreamingLLM w/ dilated	20.1	9.4	44.5	**15.5**	11.2	20.5	27.5	5.0	87.5	0.5	24.2
StreamingLLM w/ strided	17.3	8.2	27.5	14.5	11.2	19.5	27.5	4.0	2.1	1.0	13.3
InfLLM	**24.1**	7.8	45.0	6.0	11.4	19.5	32.9	**100.0**	**100.0**	1.2	34.8
Ours w/ static	19.9	8.6	43.2	3.5	8.9	20.6	25.1	92.4	96.3	0.2	31.9
Ours	20.5	**12.9**	**65.9**	7.5	**12.5**	22.3	**33.1**	**100.0**	**100.0**	**12.8**	**38.8**
Yi-9B-200K	8.2	10.6	64.2	1.0	17.3	21.3	23.4	99.8	100.0	28.8	37.5
StreamingLLM	5.4	**14.2**	38.0	**4.0**	18.8	18.8	22.3	39.2	6.1	1.6	16.8
StreamingLLM w/ dilated	5.7	4.2	15.0	0.0	18.2	0.0	2.9	0.0	0.0	0.0	4.2
StreamingLLM w/ strided	6.1	4.5	9.8	0.0	16.9	0.0	3.1	1.5	0.0	0.0	4.6
InfLLM	6.3	13.0	45.9	2.5	**21.5**	20.6	34.6	85.3	88.1	1.4	31.9
Ours w/ static	5.8	12.6	48.5	3.0	12.6	20.8	**25.1**	60.9	38.5	1.0	22.9
Ours	**7.9**	11.2	**64.2**	1.0	17.9	**24.1**	23.1	**99.5**	**100.0**	**27.6**	**37.7**

Table 3. Performance of different models and different methods on RULER (Hsieh et al., 2024) evaluated at lengths from 4k to 128k.

Methods	Claimed	Effective	4K	8K	16K	32K	64K	128K	Avg.
LLaMA-3-262K	262K	16K	97.2	91.8	87.3	80.8	77.4	72.2	84.4
StreamingLLM	-	4K	97.2	38.1	37.5	17.2	14.2	9.4	35.0
StreamingLLM w/ dilated	-	<4K	23.4	0.7	1.4	18.8	16.5	15.6	12.7
StreamingLLM w/ strided	-	<4K	2.0	0.7	0.6	0.6	0.7	1.3	1.0
InfLLM	-	4K	89.4	79.8	70.1	55.6	43.0	39.5	62.9
Ours	-	**32K**	**97.7**	**91.2**	**88.5**	**85.0**	**82.3**	**77.6**	**87.0**
Yi-200K	200K	8K	91.9	90.2	78.8	76.3	68.1	62.9	78.1
StreamingLLM	-	4K	91.9	37.8	33.9	18.6	13.0	12.8	34.3
StreamingLLM w/ dilated	-	<4K	44.8	42.8	38.5	29.8	26.8	23.9	34.4
StreamingLLM w/ strided	-	<4K	2.6	0.7	0.6	0.6	1.2	0.5	1.1
InfLLM	-	<4K	80.3	83.9	60.7	45.2	38.6	30.2	56.5
Ours	-	8K	**92.3**	**89.7**	**79.0**	**73.8**	**64.7**	**56.9**	**74.7**

RULER We evaluate MInference with the state-of-the-art long-context challenge, RULER. As shown in Table 3, our method effectively maintains the long-context performance even in complex multi-hop or aggregation tasks in RULER. It even outperforms the original full attention when the testing length beyond 32K. LLaMA-3 achieves a 32K effective context window (context with performance over 85% is considered effective (Hsieh et al., 2024)) with our method which is double than the original full attention.

5. Conclusion

This paper proposes MInference, a method that accelerates the pre-filling stage of long-context LLMs by leveraging dynamic sparse attention with spatial aggregation patterns. We categorize attention heads into three types: *A-shape*, *Vertical-Slash*, and *Block-Sparse*, and utilize a fast approximation and optimal dynamic sparse attention kernel to speed up inference. Experimental results on long-context benchmarks demonstrate that our method maintains LLM capabilities while achieving up to a 10x speedup, reducing latency from 30 mins to 3 mins per prompt for 1M token prompts on a single A100 GPU.

References

Triton implementation of the flash attention v2 algorithm. Technical report, OpenAI, 2023. URL https://triton-lang.org/main/getting-started/tutorials/06-fused-attention.html.

Abdin, M., Jacobs, S. A., Awan, A. A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Mendes, C. C. T., Chen, W., Chaudhary, V., Chopra, P., Giorno, A. D., de Rosa, G., Dixon, M., Eldan, R., Iter, D., Garg, A., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R. J., Huynh, J., Javaheripi, M., Jin, X., Kauffmann, P., Karampatziakis, N., Kim, D., Khademi, M., Kurilenko, L., Lee, J. R., Lee, Y. T., Li, Y., Liang, C., Liu, W., Lin, E., Lin, Z., Madan, P., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Rosset, C., Roy, S., Ruwase, O., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Song, X., Tanaka, M., Wang, X., Ward, R., Wang, G., Witte, P., Wyatt, M., Xu, C., Xu, J., Yadav, S., Yang, F., Yang, Z., Yu, D., Zhang, C., Zhang, C., Zhang, J., Zhang, L. L., Zhang, Y., Zhang, Y., Zhang, Y., and Zhou, X. Phi-3 technical report: A highly capable language model locally on your phone. *ArXiv*, abs/2404.14219, 2024. URL https://arxiv.org/abs/2404.14219.

Adnan, M., Arunkumar, A., Jain, G., Nair, P., Soloveychik, I., and Kamath, P. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. 2023.

Anagnostidis, S., Pavllo, D., Biggio, L., Noci, L., Lucchi, A., and Hofmann, T. Dynamic context pruning for efficient and interpretable autoregressive transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

Bairi, R., Sonwane, A., Kanade, A., C, V. D., Iyer, A., Parthasarathy, S., Rajamani, S., Ashok, B., and Shet, S. Codeplan: Repository-level coding using LLMs and planning. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. URL https://openreview.net/forum?id=d0A2pc2kFp.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *ArXiv preprint*, abs/2004.05150, 2020. URL https://arxiv.org/abs/2004.05150.

Bertsch, A., Alon, U., Neubig, G., and Gormley, M. R. Unlimiformer: Long-range transformers with unlimited length input. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=lJWUJWLCJo.

Caciularu, A., Peters, M. E., Goldberger, J., Dagan, I., and Cohan, A. Peek across: Improving multi-document modeling via cross-document question-answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1970–1989, 2023.

Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation. *ArXiv preprint*, abs/2306.15595, 2023. URL https://arxiv.org/abs/2306.15595.

Chen, Z., Wu, J., Wang, W., Su, W., Chen, G., Xing, S., Zhong, M., Zhang, Q., Zhu, X., Lu, L., et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24185–24198, 2024.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *ArXiv preprint*, abs/1904.10509, 2019. URL https://arxiv.org/abs/1904.10509.

Dai, J., Huang, Z., Jiang, H., Chen, C., Cai, D., Bi, W., and Shi, S. Sequence can secretly tell you what to discard. *ArXiv preprint*, abs/2404.15949, 2024. URL https://arxiv.org/abs/2404.15949.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024.

Dao, T. and Gu, A. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=ztn8FCR1td.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

Deng, Y., Song, Z., and Yang, C. Attention is naturally sparse with gaussian distributed input. *ArXiv preprint*, abs/2404.02690, 2024. URL https://arxiv.org/abs/2404.02690.

Ding, J., Ma, S., Dong, L., Zhang, X., Huang, S., Wang, W., Zheng, N., and Wei, F. Longnet: Scaling transformers to 1,000,000,000 tokens. *ArXiv preprint*, abs/2307.02486, 2023. URL https://arxiv.org/abs/2307.02486.

Ding, Y., Zhang, L. L., Zhang, C., Xu, Y., Shang, N., Xu, J., Yang, F., and Yang, M. LongroPE: Extending LLM context window beyond 2 million tokens. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=ONOtpXLqqw.

Dong, H., Yang, X., Zhang, Z., Wang, Z., Chi, Y., and Chen, B. Get more with LESS: Synthesizing recurrence with KV cache compression for efficient LLM inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=uhHDhVKFMW.

Fu, Y. Challenges in deploying long-context transformers: A theoretical peak performance analysis. *ArXiv preprint*, abs/2405.08944, 2024. URL https://arxiv.org/abs/2405.08944.

Fu, Y., Panda, R., Niu, X., Yue, X., Hajishirzi, H., Kim, Y., and Peng, H. Data engineering for scaling language models to 128k context. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=TaAqeo7lUh.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uNrFpDPMyo.

GLM, T., Zeng, A., Xu, B., Wang, B., Zhang, C., Yin, D., Rojas, D., Feng, G., Zhao, H., Lai, H., et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *ArXiv preprint*, abs/2406.12793, 2024. URL https://arxiv.org/abs/2406.12793.

Gradient. Llama-3 8b instruct gradient 4194k (v0.1), 2024.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *ArXiv preprint*, abs/2312.00752, 2023. URL https://arxiv.org/abs/2312.00752.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Han, C., Wang, Q., Peng, H., Xiong, W., Chen, Y., Ji, H., and Wang, S. LM-infinite: Zero-shot extreme length generalization for large language models. In Duh, K., Gomez, H., and Bethard, S. (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3991–4008, Mexico City, Mexico, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.naacl-long.222.

Ho, N., Bae, S., Kim, T., Jo, H., Kim, Y., Schuster, T., Fisch, A., Thorne, J., and Yun, S.-Y. Block transformer: Global-to-local language modeling for fast inference. *ArXiv preprint*, abs/2406.02657, 2024. URL https://arxiv.org/abs/2406.02657.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. Ruler: What's the real context size of your long-context language models? *ArXiv preprint*, abs/2404.06654, 2024. URL https://arxiv.org/abs/2404.06654.

Jacobs, S. A., Tanaka, M., Zhang, C., Zhang, M., Song, L., Rajbhandari, S., and He, Y. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *ArXiv preprint*, abs/2309.14509, 2023. URL https://arxiv.org/abs/2309.14509.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *ArXiv preprint*, abs/2310.06825, 2023a. URL https://arxiv.org/abs/2310.06825.

Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., and Qiu, L. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, 2023b.

Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2023.

Kamradt, G. Needle in a haystack - pressure testing llms, 2023.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings*

of Machine Learning Research, pp. 5156–5165. PMLR, 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.html.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Lagunas, F., Charlaix, E., Sanh, V., and Rush, A. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 10619–10629, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.829. URL https://aclanthology.org/2021.emnlp-main.829.

Li, T., Zhang, G., Do, Q. D., Yue, X., and Chen, W. Long-context llms struggle with long in-context learning. *ArXiv preprint*, abs/2404.02060, 2024a. URL https://arxiv.org/abs/2404.02060.

Li, Y., Dong, B., Guerin, F., and Lin, C. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. Snapkv: Llm knows what you are looking for before generation. *ArXiv preprint*, abs/2404.14469, 2024b. URL https://arxiv.org/abs/2404.14469.

Lieber, O., Lenz, B., Bata, H., Cohen, G., Osin, J., Dalmedigos, I., Safahi, E., Meirom, S., Belinkov, Y., Shalev-Shwartz, S., et al. Jamba: A hybrid transformer-mamba language model. *ArXiv preprint*, abs/2403.19887, 2024. URL https://arxiv.org/abs/2403.19887.

Likhosherstov, V., Choromanski, K., and Weller, A. On the expressive power of self-attention matrices. *ArXiv preprint*, abs/2106.03764, 2021. URL https://arxiv.org/abs/2106.03764.

Likhosherstov, V., Choromanski, K., and Weller, A. On the expressive flexibility of self-attention matrices. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(7):8773–8781, 2023. doi: 10.1609/aaai.v37i7.26055. URL https://ojs.aaai.org/index.php/AAAI/article/view/26055.

Lin, Z., Miao, Y., Zhang, Q., Yang, F., Zhu, Y., Li, C., Maleki, S., Cao, X., Shang, N., Yang, Y., Xu, W., Yang, M., Zhang, L., and Zhou, L. nnscaler: Constraint-guided parallelization plan generation for deep learning training. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, 2024.

Liu, H., Zaharia, M., and Abbeel, P. Ring attention with blockwise transformers for near-infinite context. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023a.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024a.

Liu, H., Yan, W., Zaharia, M., and Abbeel, P. World model on million-length video and language with ringattention. *ArXiv preprint*, abs/2402.08268, 2024b. URL https://arxiv.org/abs/2402.08268.

Liu, L., Qu, Z., Chen, Z., Tu, F., Ding, Y., and Xie, Y. Dynamic sparse attention for scalable transformer acceleration. *IEEE Transactions on Computers*, 71(12): 3165–3178, 2022.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., and Chen, B. Deja vu: Contextual sparsity for efficient LLMs at inference time. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2023b. URL https://proceedings.mlr.press/v202/liu23am.html.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024c.

Mao, Y., Ester, M., and Li, K. Iceformer: Accelerated inference with long-sequence transformers on CPUs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=6RR3wU4mSZ.

Munkhdalai, T., Faruqui, M., and Gopal, S. Leave no context behind: Efficient infinite context transformers with infini-attention. *ArXiv preprint*, abs/2404.07143, 2024. URL https://arxiv.org/abs/2404.07143.

Nawrot, P., Łańcucki, A., Chochowski, M., Tarjan, D., and Ponti, E. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=tDRYrAkOB7.

Nijkamp, E., Xie, T., Hayashi, H., Pang, B., Xia, C., Xing, C., Vig, J., Yavuz, S., Laban, P., Krause, B., Purushwalkam, S., Niu, T., Kryściński, W., Murakhovs'ka, L., Choubey, P. K., Fabbri, A., Liu, Y., Meng, R., Tu, L., Bhat, M., Wu, C.-S., Savarese, S., Zhou, Y., Joty, S., and Xiong, C. Xgen-7b technical report. *ArXiv preprint*, abs/2309.03450, 2023. URL https://arxiv.org/abs/2309.03450.

Oren, M., Hassid, M., Adi, Y., and Schwartz, R. Transformers are multi-state rnns. *ArXiv preprint*, abs/2401.06104, 2024. URL https://arxiv.org/abs/2401.06104.

Pagliardini, M., Paliotta, D., Jaggi, M., and Fleuret, F. Fast attention over long sequences with dynamic sparse flash attention. *Advances in Neural Information Processing Systems*, 36, 2024.

Pan, Z., Wu, Q., Jiang, H., Xia, M., Luo, X., Zhang, J., Lin, Q., Rühle, V., Yang, Y., Lin, C.-Y., et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*. Association for Computational Linguistics, 2024.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior. *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. URL https://api.semanticscholar.org/CorpusID:258040990.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Derczynski, L., Du, X., Grella, M., Gv, K., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Lin, J., Mantri, K. S. I., Mom, F., Saito, A., Song, G., Tang, X., Wind, J., Woźniak, S., Zhang, Z., Zhou, Q., Zhu, J., and Zhu, R.-J. RWKV: Reinventing RNNs for the transformer era. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14048–14077, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.936. URL https://aclanthology.org/2023.findings-emnlp.936.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

Reid, M., Savinov, N., Teplyashin, D., Lepikhin, D., Lillicrap, T., Alayrac, J.-b., Soricut, R., Lazaridou, A., Firat, O., Schrittwieser, J., et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *ArXiv preprint*, abs/2403.05530, 2024. URL https://arxiv.org/abs/2403.05530.

Ren, L., Liu, Y., Lu, Y., Shen, Y., Liang, C., and Chen, W. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *ArXiv preprint*, abs/2406.07522, 2024. URL https://arxiv.org/abs/2406.07522.

Ribar, L., Chelombiev, I., Hudlass-Galley, L., Blake, C., Luschi, C., and Orr, D. Sparq attention: Bandwidth-efficient LLM inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=OS5dqxmmtl.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021. doi: 10.1162/tacl_a_00353. URL https://aclanthology.org/2021.tacl-1.4.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *ArXiv preprint*, abs/1911.02150, 2019. URL https://arxiv.org/abs/1911.02150.

Shi, H., Gao, J., Ren, X., Xu, H., Liang, X., Li, Z., and Kwok, J. T. Sparsebert: Rethinking the importance analysis in self-attention. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9547–9557. PMLR, 2021. URL http://proceedings.mlr.press/v139/shi21a.html.

Sun, H., Chen, Z., Yang, X., Tian, Y., and Chen, B. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *ArXiv preprint*, abs/2404.11912, 2024a. URL https://arxiv.org/abs/2404.11912.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. Retentive network: A successor to transformer for large language models. *ArXiv preprint*, abs/2307.08621, 2023. URL https://arxiv.org/abs/2307.08621.

Sun, Y., Dong, L., Zhu, Y., Huang, S., Wang, W., Ma, S., Zhang, Q., Wang, J., and Wei, F. You only cache once: Decoder-decoder architectures for language models. *ArXiv preprint*, abs/2405.05254, 2024b. URL https://arxiv.org/abs/2405.05254.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=KzACYw0MTV.

Tay, Y., Dehghani, M., Tran, V. Q., Garcia, X., Wei, J., Wang, X., Chung, H. W., Bahri, D., Schuster, T., Zheng, S., Zhou, D., Houlsby, N., and Metzler, D. UL2: Unifying language learning paradigms. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6ruVLB727MC.

Tillet, P., Kung, H.-T., and Cox, D. 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*, pp. 10–19, 2019.

Tworkowski, S., Staniszewski, K., Pacek, M., Wu, Y., Michalewski, H., and Miłoś, P. Focused transformer: Contrastive training for context scaling. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=s1FjXzJ0jy.

Wan, Z., Wu, Z., Liu, C., Huang, J., Zhu, Z., Jin, P., Wang, L., and Yuan, L. Look-m: Look-once optimization in kv cache for efficient multimodal long-context inference. *ArXiv preprint*, abs/2406.18139, 2024. URL https://arxiv.org/abs/2406.18139.

Wang, H., Zhang, Z., and Han, S. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 97–110. IEEE, 2021.

Weng, L. Llm-powered autonomous agents. *lilianweng.github.io*, 2023. URL https://lilianweng.github.io/posts/2023-06-23-agent/.

Xiao, C., Zhang, P., Han, X., Xiao, G., Lin, Y., Zhang, Z., Liu, Z., Han, S., and Sun, M. Infllm: Unveiling the intrinsic capacity of llms for understanding extremely long sequences with training-free memory. *ArXiv preprint*, abs/2402.04617, 2024a. URL https://arxiv.org/abs/2402.04617.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024b.

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., Xu, J., Zhou, J., Bai, J., He, J., Lin, J., Dang, K., Lu, K., Chen, K., Yang, K., Li, M., Xue, M., Ni, N., Zhang, P., Wang, P., Peng, R., Men, R., Gao, R., Lin, R., Wang, S., Bai, S., Tan, S., Zhu, T., Li, T., Liu, T., Ge, W., Deng, X., Zhou, X., Ren, X., Zhang, X., Wei, X., Ren, X., Fan, Y., Yao, Y., Zhang, Y., Wan, Y., Chu, Y., Liu, Y., Cui, Z., Zhang, Z., and Fan, Z. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024. URL https://arxiv.org/abs/2407.10671.

Young, A., Chen, B., Li, C., Huang, C., Zhang, G., Zhang, G., Li, H., Zhu, J., Chen, J., Chang, J., et al. Yi: Open foundation models by 01. ai. *ArXiv preprint*, abs/2403.04652, 2024. URL https://arxiv.org/abs/2403.04652.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. Big bird: Transformers for longer sequences. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html.

Zhang, X., Chen, Y., Hu, S., Xu, Z., Chen, J., Hao, M. K., Han, X., Thai, Z. L., Wang, S., Liu, Z., et al. ∞bench: Extending long context evaluation beyond 100k tokens. *ArXiv preprint*, abs/2402.13718, 2024a. URL https://arxiv.org/abs/2402.13718.

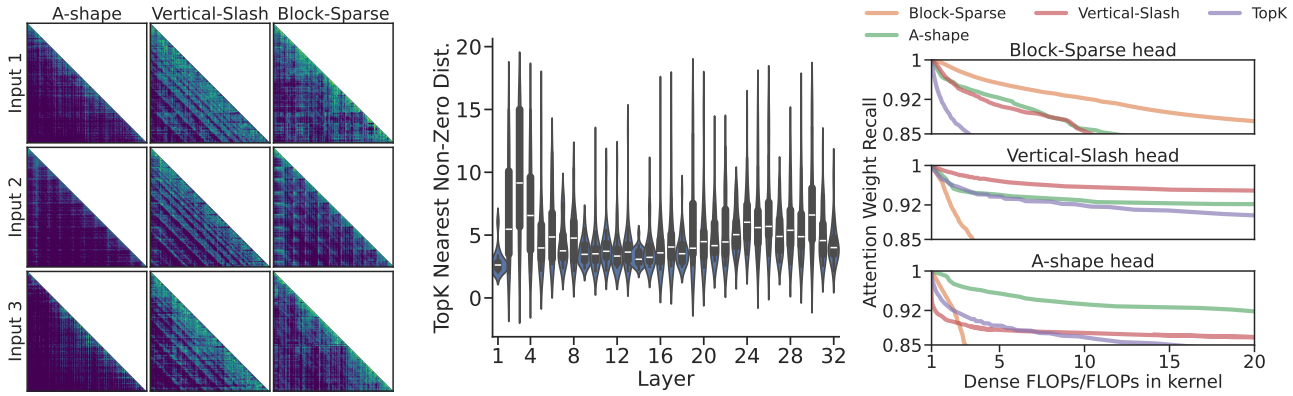
Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024b.

Zheng, N., Jiang, H., Zhang, Q., Han, Z., Ma, L., Yang, Y., Yang, F., Zhang, C., Qiu, L., Yang, M., et al. Pit: Optimization of dynamic sparse deep learning models via permutation invariant transformation. In *Proceedings of*

the 29th Symposium on Operating Systems Principles, pp. 331–347, 2023.

Zimerman, I., Ali, A., and Wolf, L. A unified implicit attention formulation for gated-linear recurrent sequence models. *ArXiv preprint*, abs/2405.16504, 2024. URL https://arxiv.org/abs/2405.16504.

A. Additional Motivation Details



(a) Attention patterns (b) Attention is spatial clustering (c) Attention pattern recall

Figure 4. (a) Visualization of attention weights from different attention heads. For different prompts and tasks, the pattern of the same head is relatively consistent, but the sparse indices are dynamically changing.(b) Distance of the top-10 nearest non-zero element in the attention matrix. (c) Attention recall distribution using our identified patterns, where FLOPs in the kernel refer to the real FLOPs required for sparse attention computing using on GPUs. Here, a 1x64 block size is used for the *Vertical-Slash* pattern, and a 64x64 block size is used for others on GPUs. All visualization are based on LLaMA-3-8B-Instruct-262K.

B. Additional Methodology Details

B.1. Problem Formulation

When accelerating the pre-filling stage of long-context LLMs with sparse attention computing, the attention matrix can be formulated as follows:

$$\boldsymbol{A}(\boldsymbol{M}) = \text{Softmax}(\frac{1}{\sqrt{d}}\boldsymbol{Q}\boldsymbol{K}^\top - c(1-\boldsymbol{M})), \tag{1}$$

where $M_{i,j} \in \{0,1\}$ represents the dynamic sparse mask for item i,j of the attention matrix. Here, c is a large constant, such as 1e5, ensuring that the less important attention weights for which $M_{i,j} = 0$ have values approaching zero after the softmax, i.e., $A_{i,j} \approx 0$.

The goal of the dynamic sparse attention system is to achieve greater speedup with minimal overhead while retaining as much of the attention weights as possible. Formally, this can be expressed as:

$$\begin{aligned} &\min \quad |\boldsymbol{A}(\boldsymbol{M}) - \boldsymbol{A}_{\text{dense}}|, \\ &\min \quad t_{\text{sparse}}(\boldsymbol{M}) + t_{\text{overhead}}(\boldsymbol{M}), \end{aligned} \tag{2}$$

where t_{sparse} and t_{overhead} represent the time spent on dynamic sparse attention computation and estimation of the approximate dynamic sparse pattern, respectively.

B.2. Kernel-Aware Optimal Sparse Pattern Search Algorithm

B.3. Sparsity Indices Approximation and Dynamic Sparse Attention Calculation Algorithm

C. Experiment Details

C.1. Dataset Details

We use the provided metrics and scripts from the following benchmarks for evaluation.

InfiniteBench (Zhang et al., 2024a) includes 10 tasks designed to test various aspects of long-context processing. Specifically, these tasks cover entire novel summarization, open-form question answering based on novels, multiple-choice

Algorithm 1 Kernel-Aware Sparse Pattern Search

Input: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{S \times d_h}$, patterns p, search space ρ, target FLOPs t, initialized search space σ

 # Build kernel-aware search space
for $i \leftarrow 1$ to $|\sigma|$ **do**
  $t_i \leftarrow$ FLOPs_in_kernel(σ_i)
  while $|t_i - t| > \epsilon$ **do**
    $\sigma_i \leftarrow$ ChangeSpace(σ_i, p_i)
    $t_i \leftarrow$ FLOPs_in_kernel(σ_i)
  end while
  $\rho \leftarrow \rho \cup \sigma_i$
end for

 # Search for optimal head pattern
$p_{\text{best}} \leftarrow \phi$
$\boldsymbol{y} \leftarrow \text{Softmax}(\boldsymbol{Q}\boldsymbol{K}^\top / \sqrt{d})$
for $i \leftarrow 1$ to $|\rho|$ **do**
  $\boldsymbol{y}_i \leftarrow \text{SparseAttention}(\boldsymbol{Q}\boldsymbol{K}^\top / \sqrt{d}, \rho_i)$
  $p_{\text{best}} \leftarrow \text{argmin}(\boldsymbol{y}_i - \boldsymbol{y}, p_{\text{best}})$
end for
return p_{best}

question answering on novels, question answering on long drama scripts, question answering on Chinese texts, debugging large code repositories, identifying the largest/smallest number in arrays, and retrieval tasks with varying pattern lengths. The average token length for these tasks is 214k, and they include 3,992 examples.

RULER (Hsieh et al., 2024) is a recent synthetic benchmark suite for long-context evaluation with 13 complex tasks across four categories. The retrieval category includes Single Needle-in-a-Haystack (S-NIAH), where a single key-value pair is inserted into noisy text, and the model must retrieve the value. Multi-keys Needle-in-a-Haystack (MK-NIAH) involves multiple keys, and the model retrieves one specific value among hard distractors. The Multi-values Needle-in-a-Haystack (MV-NIAH) task requires retrieving all values associated with a single key, while the Multi-queries Needle-in-a-Haystack (MQ-NIAH) task involves retrieving values for multiple keys. The Multi-hop Tracing category includes Variable Tracking (VT), where the model traces and returns all variable names pointing to the same value through variable bindings. The aggregation category introduces Common Words Extraction (CWE), where the model identifies the top-K common words from a mixture of common and uncommon words, and Frequent Words Extraction (FWE), where the model identifies the most frequent words from a Zeta distribution. The Question Answering (QA) category extends existing short-context QA datasets by adding distracting paragraphs, challenging the model to answer questions based on relevant information surrounded by distractors. These tasks provide a comprehensive evaluation of long-context modeling capabilities, covering multi-hop reasoning, aggregation, and complex question answering. Following (Hsieh et al., 2024), we test models on 4K, 8K, 16K, 32K, 64K, and 128K context lengths, including 2,600 examples per length.

Needle In A Haystack task (Kamradt, 2023) evaluates the performance of retrieval-augmented generation (RAG) systems by embedding specific, targeted information (the "needle") within a large, complex body of text (the "haystack"). The test assesses a language model's ability to identify and utilize this specific piece of information amidst a vast amount of data. Both RULER and the needle test iterate over various context lengths and document depths (where the ground-truth is placed in the prompt) to measure the long-context performance. Here we scale the Needle In A Haystack task to 1M context length, including 750 examples.

PG-19 (Rae et al., 2020) The perplexity on long text is also often used by researchers to evaluate the language modeling performance of long-context LLMs. PG-19 is a suitable test set for this task, as it includes texts as long as 500K tokens. Perplexity is used as the metric indicating how well a model predicts the next token in a sequence. Our experiments are conducted on 1,000 random samples from PG-19 that are longer than 100K tokens.

Algorithm 2 Vertical-Slash Head

Input: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{S \times d_h}$, $k_v, k_s \in \mathbb{N}$

 # Approximate vertical and slash pattern (last_q = 64)

$\widehat{\boldsymbol{A}} \leftarrow \text{softmax}\left(\boldsymbol{Q}_{[-\text{last\_q}:]}\boldsymbol{K}^\top/\sqrt{d} + \boldsymbol{m}_{\text{casual}}\right)$

 # Indices of top k_v vertical line, sum in vertical

$\boldsymbol{i}_v \leftarrow \text{argtopk}\left(\text{sum}_v(\widehat{\boldsymbol{A}}), k_v\right)$

 # Indices of top k_s slash line, sum in slash

$\boldsymbol{i}_s \leftarrow \text{argtopk}\left(\text{sum}_s(\widehat{\boldsymbol{A}}), k_s\right)$

 # Build sparse attention index

$\boldsymbol{i}_{vs} \leftarrow \text{sparseformat}(\boldsymbol{i}_v, \boldsymbol{i}_s)$

 # Final dynamic sparse attention scores (only index block)

$\boldsymbol{A} \leftarrow \text{softmax}\left(\text{sparse}(\boldsymbol{Q}\boldsymbol{K}^\top, \boldsymbol{i}_{vs})/\sqrt{d}\right)$

 # Sparse mixed scores and values

$\boldsymbol{y} \leftarrow \text{sparse}(\boldsymbol{A}\boldsymbol{V}, \boldsymbol{i}_{vs})$

return $\boldsymbol{y}$

Algorithm 3 Block-Sparse Head

Input: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{S \times d_h}$, $k_b \in \mathbb{N}$

 # Approximate block-sparse pattern (block_size = 64)

$\widehat{\boldsymbol{Q}} \leftarrow \text{MeanPooling}(\boldsymbol{Q}, block\_size)$

$\widehat{\boldsymbol{K}} \leftarrow \text{MeanPooling}(\boldsymbol{K}, block\_size)$

$\widehat{\boldsymbol{A}} \leftarrow \text{softmax}\left(\widehat{\boldsymbol{Q}}\widehat{\boldsymbol{K}}^\top/\sqrt{d} + \boldsymbol{m}_{\text{casual}}\right)$

 # Indices of top k blocks

$\boldsymbol{i}_b \leftarrow \text{argtopk}\left(\widehat{\boldsymbol{A}}, k_b\right)$

 # Build sparse attention index

$\boldsymbol{i}_b \leftarrow \text{sparseformat}(\boldsymbol{i}_b)$

 # Final dynamic sparse attention scores (only index block)

$\boldsymbol{A} \leftarrow \text{softmax}\left(\text{sparse}(\boldsymbol{Q}\boldsymbol{K}^\top, \boldsymbol{i}_b)/\sqrt{d}\right)$

 # Sparse mixed scores and values

$\boldsymbol{y} \leftarrow \text{sparse}(\boldsymbol{A}\boldsymbol{V}, \boldsymbol{i}_b)$

return $\boldsymbol{y}$

C.2. Baselines

We include five training-free sparse attention approaches as our baselines: 1) StreamingLLM (Xiao et al., 2024b), which corresponds to the *A-shape* pattern. We use 1k global tokens and 4k local windows in all our experiments; 2) StreamingLLM w/ dilated (Beltagy et al., 2020), which sets dilated local windows with intervals in the local windows direction. We use 1k global tokens and 8k dilated attention windows with an interval of 1; 3) StreamingLLM w/ strided (Child et al., 2019), which retains local windows while adding dilated attention. We use 1k global tokens, 2k local windows, and 4k dilated attention windows with an interval of 1; 4) InfLLM (Xiao et al., 2024a), which uses a memory unit to process streaming long sequences. Following the paper, we set 128 global tokens and 8k local windows in all experiments; 5) Ours w/ static, which utilizes static sparse indices in the *Vertical-Slash* and *Block-Sparse* heads. For all baselines, we perform sparse computation only during the pre-filling stage, while retaining dense computation during the decoding stage.

C.3. Implementation Details

Our experiments are based on a number of state-of-the-art long-context LLMs: 1) LLaMA-3-8B-Instruct-262k[†] is a LLaMA-3 variant with further NTK-aware interpolation and minimal fine-tuning with Ring Attention, which achieved SOTA results on long-context assessments such as the Needle In A Haystack test; 2) LLaMA-3-8B-Instruct-1048k[‡] is similar to LLaMA-3-8B-Instruct-262k, but supports context lengths up to 1M tokens; 3) Yi-9B-200K (Young et al., 2024) is a SOTA LLM that balances long-context performance with general capabilities; 4) Phi-3-Mini-128K (Abdin et al., 2024) a small but powerful language model that offers capabilities equivalent to models ten times its size with up to 128K context window powered by LongRoPE (Ding et al., 2024); 5) Qwen2-7B-128K (Yang et al., 2024) is a recently release update of Qwen series model with up to 128K context window that achieve superior or comparable performance compared to LLaMA-3; 6) GLM-4-9B-1M (GLM et al., 2024) has been improved from its predecessor in terms of a 1M context window, performance on downstream tasks and inference efficiency. To guarantee stable results, we use greedy decoding in all tests. Our kernel implementations are developed and optimized based on the dynamic sparse compiler PIT (Zheng et al., 2023) in the Triton language (Tillet et al., 2019). The latency experiments are done on a single Nvidia A100 GPU using bfloat16. We provide a simple custom implementation of attention in PyTorch, building on FlashAttention and Triton.

We set the target FLOPs t to be the same as 1k global tokens and 4k local window tokens in the *A-shape* pattern. The step size of ChangeSpace is set to 50, with the corresponding search space shown in Table 4. Additionally, we use only one sample as our validation set from KV retrieval synthetic data with 30k token inputs, which exhibits strong generalization and stability across different lengths and domains. The search time is approximately 15 minutes on a single

[†]https://huggingface.co/gradientai/Llama-3-70B-Instruct-Gradient-262k

[‡]https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k

A100. Additionally, we use the same optimal sparse pattern configuration for both the LLaMA-3-8B-Instruct-262K model and the LLaMA-3-8B-Instruct-1M model. The specific distribution is shown in Fig. 9.

Table 4. Kernal-aware optimal head pattern search space. In this context, *A-shape* represents the global tokens and local window number, *Vertical-Slash* represents the Top-K number of vertical and diagonal lines, and *Block-Sparse* represents the Top-K number of blocks retained.

Patterns	Search Space
A-shape	$\{(1024, 4096)\}$
Vertical-Slash	$\{(30, 2048), (100, 1800), (500, 1500), (3000, 200)\}$
Block-Sparse	$\{100\}$

C.4. Single A100 Implementation Details

The original PyTorch implementation[§] of the LLaMA model causes an out-of-memory error on a single A100 (80G) when the prompt exceeds 50k tokens. To enable running 1M prompt inference on a single A100, we implemented the following optimizations:

1. **Tensor Splitting**: We split the Attention by head and the MLP by sequence dimension. In long-context scenarios, where computation is the bottleneck, this splitting keeps GPU utilization at 100%, and the overhead of splitting is negligible;
2. **Reduction of Intermediate Variables**: We minimized the allocation of intermediate variables by removing the Attention Mask and implementing the causal mask logic within the kernel;
3. **Elimination of Unnecessary Computations**: In long-context scenarios, only the logits corresponding to the last token in the prompt phase are meaningful. Thus, we only retain the computation of the LM Head Linear layer for the last token.

C.5. Kernel Implementation

C.5.1. Block-Sparse Flash Attention

Our *Block-Sparse* kernel implementation is based on the Triton version of the FlashAttention kernel (tri, 2023). With the selected block index as an additional input, each thread block loops through the top-K blocks in a row. As discussed in FlashAttention (Dao, 2024), the latency of the block-sparse FlashAttention kernel is linearly related to the number of blocks, and the speedup ratio (compared to the dense FlashAttention kernel) is approximately as,

$$s_p = \frac{S}{2B \times k_b} \tag{3}$$

C.5.2. Vertical-Slash Attention

The *Vertical-Slash* attention includes two custom kernels: the *Vertical-Slash* sparse index kernel and the *Vertical-Slash* sparse FlashAttention kernel.

The *Vertical-Slash* sparse index kernel in Algorithm 4 builds the index for each row of blocks. Since a slash line segment can be masked by a square block, our attention mask is a mix of blocks and columns, as shown in Fig. 5. We apply a point-range two-way merge algorithm where vertical indexes are treated as points and slash indexes are converted to ranges given the row index. The output consists of two parts: merged ranges and separate column indexes, where the ranges are represented by block indexes. The time complexity to build an index for a row is $O(k_v + k_s)$.

The *Vertical-Slash* sparse FlashAttention kernel in Algorithm 5 is a mix of the block-sparse attention kernel and the PIT (Zheng et al., 2023) sparse attention kernel. PIT is a technology that loads sparse data into dense compute blocks via a Permutation Invariant Transformation. A thread block first loops through the block indexes as described in the previous section (block part) and then loops through the column indexes grouped by block size (PIT part). The latency of this hybrid kernel is linearly related to the total area of blocks and columns.

[§]https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py

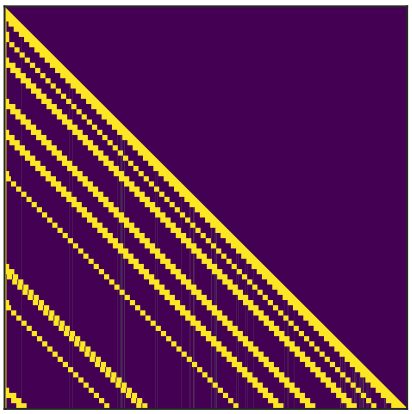


Figure 5. The dynamic sparse mask in the kernel of the vertical-slash pattern schematic using LLaMA-3-8B in the summarization task (Zhang et al., 2024a), where the yellow areas indicate the parts actually involved in computation. The slash lines are covered using 64×64 block sizes, while the vertical lines are covered using 1×64 block sizes.

Table 5. Performance of different methods with GLM-4-9B-1M (GLM et al., 2024) on InfiniteBench (Zhang et al., 2024a).

Methods	En.Sum	En.QA	En.MC	En.Dia	Zh.QA	Code.Debug	Math.Find	Retr.PassKey	Retr.Number	Retr.KV	Avg.
GLM-4-9B-1M	28.3	9.7	68.6	39.5	12.1	29.4	38.9	100.0	100.0	41.0	46.7
StreamingLLM	27.7	6.4	40.2	12.5	10.8	27.7	21.1	97.1	25.6	0.6	27.0
InfLLM	28.0	7.3	45.0	14.0	10.7	27.9	**39.4**	98.0	**100.0**	2.6	37.3
Ours	**28.8**	**9.6**	**68.6**	**38.5**	**12.0**	**30.7**	39.1	**100.0**	**100.0**	**43.0**	**47.0**

D. Additional Experiment Results

D.1. InfiniteBench

D.2. RULER

D.3. Language Modeling

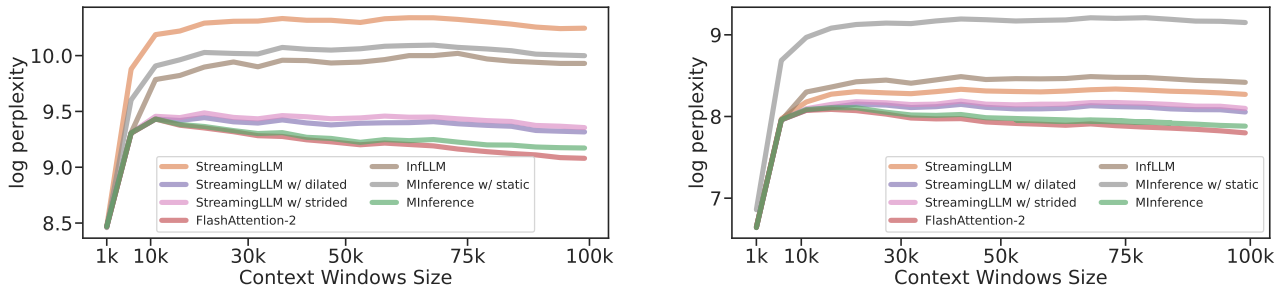
Following the approach of StreamingLLM (Xiao et al., 2024b) and H2O (Zhang et al., 2024b), we evaluate our methods against baselines on the language modeling task based on the PG-19 dataset (Rae et al., 2020). As shown in 6, our method yields best results compared to other sparse approaches, and exhibits minimal divergence compared to the full attention baseline. For prompts of 100K token, our perplexity is only 0.2 higher than the full attention, but lower than StreamingLLM for 0.25 and 0.75 on the Yi-9B-200K and LLaMA-3-262K models respectively.

D.4. Needle In A Haystack

Comparing Fig. 1a to 7, our method effectively retains the ability to process information at different positions across various context windows, ranging from 1k to 1M tokens. In contrast, methods like StreamingLLM (Xiao et al., 2024b) and InfLLM (Xiao et al., 2024a), while effective in reducing latency, experience a rapid decline in performance once the critical

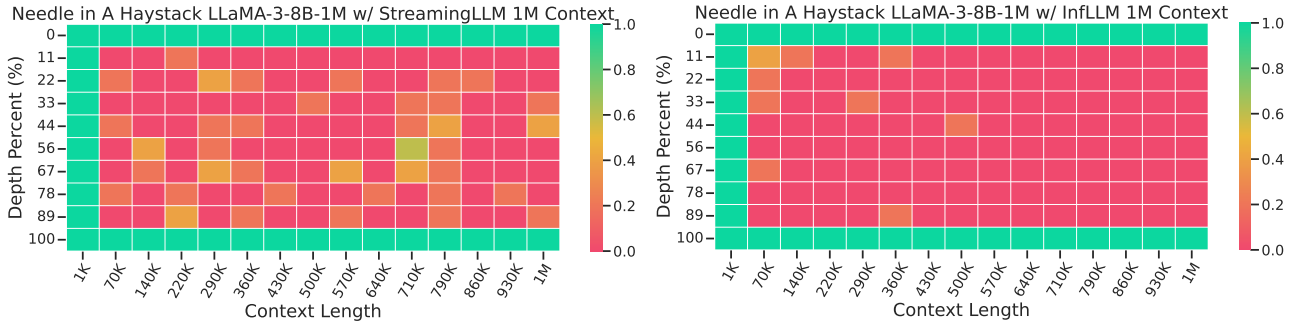
Table 6. Performance (%) of different models and different methods on RULER (Hsieh et al., 2024) evaluated at lengths from 4k to 128k.

Methods	Claimed	Effective	4K	8K	16K	32K	64K	128K	Avg.
GLM-4-9B-1M	1M	64K	93.8	91.6	89.3	87.4	85.2	80.8	88.0
StreamingLLM	-	4K	93.8	66.9	58.5	51.4	45.9	39.1	59.3
InfLLM	-	8K	**94.7**	89.5	76.4	66.5	56.8	53.5	72.9
Ours	-	64K	94.6	**93.1**	**91.0**	**89.6**	**85.5**	**84.0**	**89.6**



(a) LLaMA-3-8B-Instruct-262K (b) Yi-9B-200K

Figure 6. Perplexity results on PG-19 (Rae et al., 2020) using different models and methods.



(a) StreamingLLM (b) InfLLM

Figure 7. Needle In A Haystack (Kamradt, 2023) results using LLaMA-3-8B-Instruct-1M (Gradient, 2024).

information exceeds the range of global tokens and local windows.

In addition to the Needle In A Haystack results for LLaMA-3-Instruct-1M shown in §4, we also present the results for GLM-4-9B-1M, Yi-9B-200K, Phi-3-Mini-128K, and Qwen2-7B-128K, shown in Fig. 12. Compared to Full Attention, using MInference has minimal impact on the ability to understand semantic information across different context windows and needle depths. There is even a slight performance improvement around the 100k context length using Yi-9B-200K and Phi-3-Mini-128K.

D.5. Ablation Study

To evaluate the contributions of different components in MInference, we introduce six variants for the ablation study: (1) Ours w/ static, which uses a static sparse mask in the *Vertical-Slash* and *Block-Sparse* patterns; (2) Ours w/ only A-shape, which is equivalent to StreamingLLM; (3) Ours w/ only block-sparse, which uses only the *Block-Sparse* pattern in the dynamic sparse calculation. (4) Ours w/ only vertical-slash, which uses only the *Vertical-Slash* pattern in the dynamic sparse calculation. (5) Ours w/ only vertical, which only uses vertical lines and the top-1 slash line in *Vertical-Slash* pattern. (6) Ours w/ only slash, which only uses slash lines and the top-1 vertical line in *Vertical-Slash* pattern. The corresponding top-K quantities are set after converting based on FLOPs in kernel.

Table 7. Performance of different ablation methods using LLaMA-3-8B-Instruct-262K on InfiniteBench (Zhang et al., 2024a).

Methods	En.Sum	En.QA	En.MC	En.Dia	Zh.QA	Code.Debug	Math.Find	Retr.PassKey	Retr.Number	Retr.KV	Avg.
Ours	20.5	12.9	65.9	7.5	12.5	22.3	33.1	100.0	100.0	12.8	38.8
Ours w/ only block-sparse	12.4	3.4	5.7	6.0	3.1	12.2	24.0	59.5	60.3	0.0	18.7
Ours w/ only vertical-slash	19.6	12.0	62.1	9.5	11.7	21.6	29.1	100.0	100.0	5.0	37.1

Tables 2, 3, and 7 present the ablation results. It first proves that using static indices significantly degrades LLM performance, especially in highly dynamic tasks like KV retrieval, where accuracy nearly drops to zero. This highlight the necessity of

our dynamic strategy and the effectiveness of our dynamically built sparse indices. Additionally, remove any pattern from the three leads to varying degrees of performance degradation. Specifically, "only A-shape" can only capture information within local windows. The "only block-sparse" variant using only the *BS* pattern, also results in significant performance declines. On the other hand, "only vertical-slash" manages to preserve most of the performance due to its balance between dynamicity and the StreamingLLM pattern, but still fall behind the full version of our method. And using only vertical lines results in a significant performance drop, especially in retrieval tasks, where performance is similar to only using block-sparse. In contrast, using only slash lines retains most of the performance, but in highly dynamic tasks such as KV retrieval, performance further decreases, with an average performance drop of 2.9% compared to Ours.

D.6. Latency and Latency Breakdown

Fig. 1b and 8 shows the latency and breakdown of MInference across different context windows on a single A100. At 100K, 300K, 500K, and 1M tokens, our method achieves speedups of 1.8×, 4.1×, 6.8×, and 10×, respectively. It reduces the pre-filling latency from 30 mins to 3 mins on a single A100 for a prompt of 1M token. By further utilizing tensor parallel (Lin et al., 2024) and context parallel (Liu et al., 2023a; Jacobs et al., 2023), this latency can be reduced to 40 seconds on 8x A100 GPUs. This significantly lowers the deployment cost of long-context LLMs and enhances user experience. And since our kernel is implemented based on Triton, it can be easily ported to other devices and achieve similar speedups, such as on the H100. Additionally, analyzing the latency breakdown, we found about 5%-20% of the overhead is spent on dynamic sparse index building, while the remaining time is spent on dynamic sparse calculation.

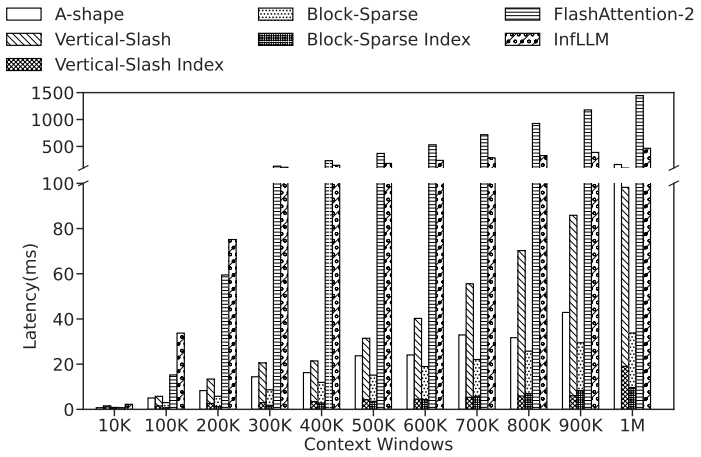


Figure 8. The latency breakdown of a single attention kernel for three patterns and FlashAttention (Dao, 2024) across different context windows in a single A100, including the index time for dynamic sparse approximation and building dynamic sparsity. At 10k tokens, the latency of the four kernels is very close and all are less than 1ms. At 1M tokens, the latency for A-shape is 164ms.

Fig. 8 shows the micro-benchmark results of the three attention patterns proposed in this paper, as well as FlashAttention. It can be seen that Vertical-Slash is the slowest among the three patterns, but it still achieves a 13x speedup compared to FlashAttention under 1M context windows. A-shape is slightly faster than Vertical-Slash, but at 1M, A-shape is 50% slower than Vertical-Slash. Block-Sparse is the fastest, achieving a 30x speedup over FlashAttention under 1M context windows. The estimation and index-building time for the dynamic sparse pattern accounts for approximately 5%-15% and 25% of the total time for Vertical-Slash and Block-Sparse patterns, respectively. The index-building overhead is higher for Block-Sparse mainly due to the time-consuming MeanPooling and block-level matmul computations. Additionally, the memory overhead for sparse indexing is relatively small, remaining within 160MB for a LLaMA-3-8B model in 1M context.

D.7. Integrate with KV cache compression methods

We also combined MInference with a state-of-the-art KV cache compression method SnapKV (Li et al., 2024b), as shown in Table 8. This proves our method is compatible with KV cache compression techniques. For most tasks, performance remains nearly unchanged, with the average score even showing a slight increase, which further demonstrates the potential practical value of our method as an optimization for serving long-context LLMs.

Table 8. Performance of different methods on InfiniteBench (Zhang et al., 2024a) using SnapKV (Li et al., 2024b) in the decoding stage.

Methods	En.Sum	En.QA	En.MC	En.Dia	Zh.QA	Code.Debug	Math.Find	Retr.PassKey	Retr.Number	Retr.KV	Avg.
LLaMA-3 w/ SnapKV	18.0	**11.8**	65.5	2.5	12.0	21.3	26.6	**100.0**	**100.0**	1.8	36.0
Ours w/ SnapKV	**18.9**	11.7	**66.4**	**6.5**	**12.1**	**21.8**	**33.1**	**100.0**	**100.0**	**2.0**	**37.3**

E. Related Works

Sparse Attention Due to the quadratic complexity of the attention mechanism, many previous works have focused on sparse attention to improve the efficiency of Transformers. These methods include static sparse patterns, cluster-based sparse approaches, and dynamic sparse attention. Static sparse patterns include techniques such as sliding windows (Jiang et al., 2023a; Abdin et al., 2024), dilated attention (Child et al., 2019; Shi et al., 2021; Ding et al., 2023), and mixed sparse patterns (Beltagy et al., 2020; Zaheer et al., 2020; Lagunas et al., 2021). Cluster-based sparse methods include hash-based (Kitaev et al., 2020) and kNN-based (Roy et al., 2021; Nawrot et al., 2024) methods. All of the above methods require pre-training the model from scratch, which makes them infeasible to be directly used as a plugin for reay-to-use LLMs. Recently, there has been work (Dao & Gu, 2024; Zimerman et al., 2024) to unify state space models (Gu et al., 2022; Gu & Dao, 2023; Dao & Gu, 2024), and linear attention (Katharopoulos et al., 2020; Sun et al., 2023) into structured masked attention. Additionally, some works (Wang et al., 2021; Liu et al., 2022; Ribar et al., 2024) leverage the dynamic nature of attention to predict sparse patterns dynamically. However, these approaches often focus on low-rank hidden states during the dynamic pattern approximation or use post-statistical methods to obtain the sparse mask, which introduce substantial overhead in the estimation step, making them less useful for long-context LLMs.

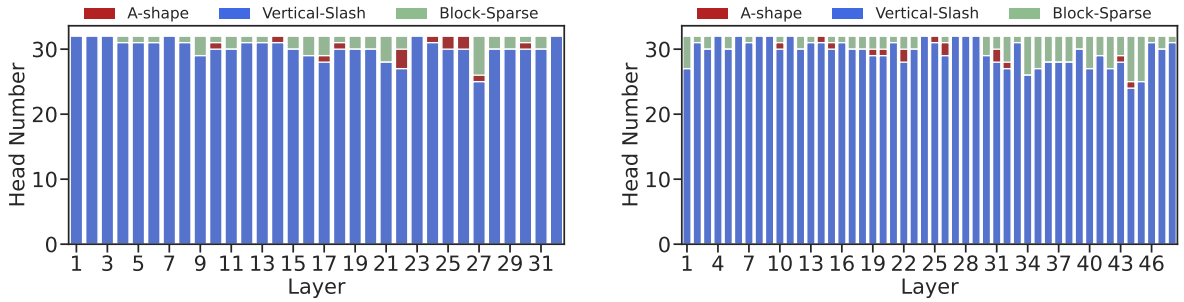
Scaling Context Windows of LLMs Recent research has focused on expanding the context window of pre-trained LLMs, that enables LLMs to handle more complex real-life applications (Jimenez et al., 2023; Park et al., 2023). These methods can be categorized into: 1) Staged pre-training (Nijkamp et al., 2023; Fu et al., 2024); 2) Modifying or interpolating position embeddings (Press et al., 2022; Chen et al., 2023; Peng et al., 2024; Ding et al., 2024); 3) Utilizing external memory modules for context storage (Bertsch et al., 2023; Tworkowski et al., 2023; Xiao et al., 2024a); 4) Expanding computations across multiple devices in a distributed manner (Liu et al., 2023a). However, these methods do not alleviate the high inference costs in long-context processing.

Long-Context LLM Inference Recent studies (Fu, 2024) have tackled the high computational cost of attention and substantial KV cache storage in long-context scenarios from two angles: pre-filling and decoding. Pre-filling optimizations are primarily categorized as State Space Models (Gu et al., 2022; Gu & Dao, 2023), linear attention methods (Sun et al., 2023; Peng et al., 2023), memory-based methods (Munkhdalai et al., 2024), hybrid methods (Lieber et al., 2024; Ho et al., 2024; Ren et al., 2024), and prompt compression methods (Li et al., 2023; Jiang et al., 2023b; 2024; Pan et al., 2024). However, these approaches require training from scratch or additional overhead and are difficult to implement directly in pretrained long-context LLMs. Recently, some studies (Mao et al., 2024; Xiao et al., 2024a) have focused on using kNN or cluster-based sparse attention to accelerate LLM inference. However, these methods often lead to reduced accuracy, limited speedup, or are restricted to CPU scenarios.

In contrast, optimizations for the decoding stage are divided into: 1) Reusing attention KV to reduce KV cache storage (Shazeer, 2019; Ainslie et al., 2023; Sun et al., 2024b; DeepSeek-AI, 2024); 2) Static KV cache compression patterns (Xiao et al., 2024b; Han et al., 2024); 3) Dynamic KV cache compression patterns, including completely discarding the KV cache after compression (Anagnostidis et al., 2024; Zhang et al., 2024b; Liu et al., 2024c; Ge et al., 2024; Oren et al., 2024), and offloading-based methods (Ribar et al., 2024; Li et al., 2024b; Dai et al., 2024); 4) Cluster-based KV cache compression methods (Nawrot et al., 2024; Tang et al., 2024); 5) Methods for restoring performance loss due to KV cache compression (Adnan et al., 2024; Dong et al., 2024); 6) Hierarchical speculative decoding methods (Sun et al., 2024a). Nevertheless, these methods do not address the heavy computational burden of the attention in the pre-filling stage.

F. Pattern Distribution

Fig. 9 shows the distribution of the optimal head configuration obtained through our search. Firstly, most of the patterns are the *Vertical-Slash* pattern (>90%). However, according to the ablation study, using only the *Vertical-Slash* pattern significantly impacts performance in highly dynamic tasks like KV retrieval. Secondly, the *Block-Sparse* pattern is primarily distributed in several intermediate to later layers, while the *A-shape* pattern is found in the middle layers. Although the



(a) LLaMA-3-8B-Instruct-262K/1M

(b) Yi-9B-200K

Figure 9. Distribution of three sparse head patterns in different models. We use the same optimal sparse pattern configuration for both LLaMA-3-8B-Instruct-262K and LLaMA-3-8B-Instruct-1M.

optimal patterns vary slightly across different models, they generally align with these observations.

Additionally, we used the same configuration for two versions of LLaMA in our experiments, and the results show that the 1M model also performs very well, with nearly perfect results in the Needle In A Haystack task. This demonstrates the generalizability of the optimal sparse pattern.

G. Sparsity in Kernel Distribution

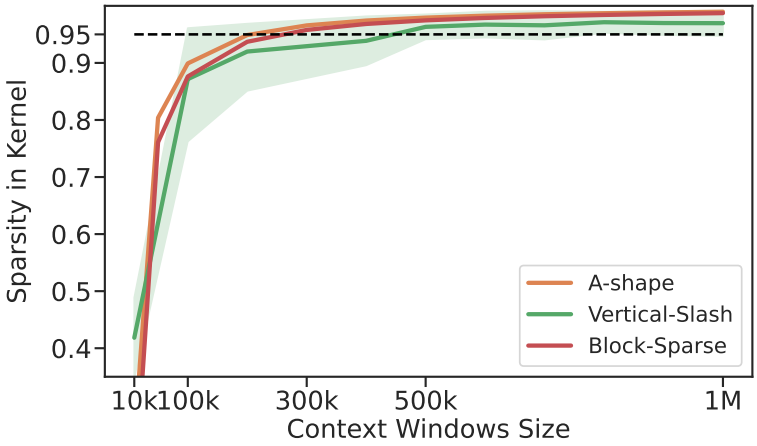


Figure 10. The distribution of sparsity in the kernel across different context windows refers to the proportion of the kernel that is actually computed after block coverage, compared to the sparsity rate when using FlashAttention with a causal mask.

As shown in Fig. 10, the sparsity distribution of the three patterns during the actual kernel computation process is displayed. It can be seen that when the context windows exceed 200k, the actual sparsity of all three patterns surpasses 90%. Even considering a 20% index-building overhead, this ensures that the kernel achieves a speedup of over 8×. Furthermore, when the context windows exceed 500k, the sparsity relative to FlashAttention exceeds 95%, with a theoretical speedup of over 15×.

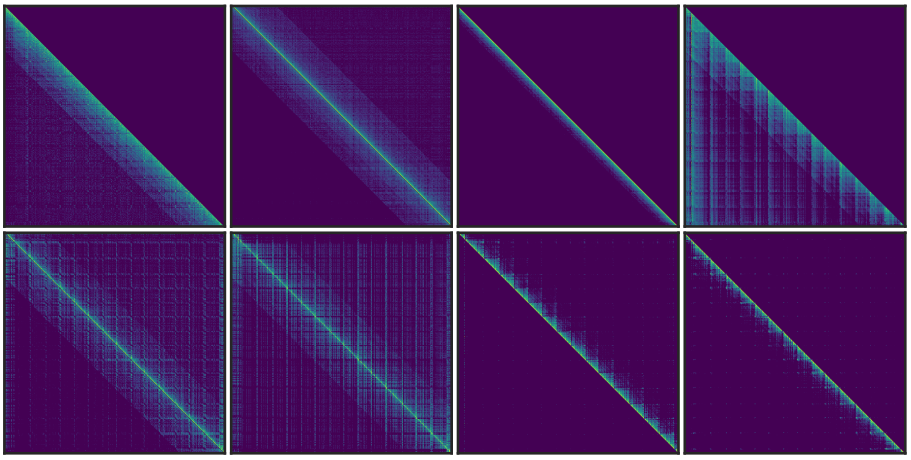


Figure 11. The sparse pattern in T5-style Encoder Attention using Flan-UL2 (Tay et al., 2023) on the Summarization (Zhang et al., 2024a).

H. Does This Dynamic Sparse Attention Pattern Exist Only in Auto-Regressive LLMs or RoPE-Based LLMs?

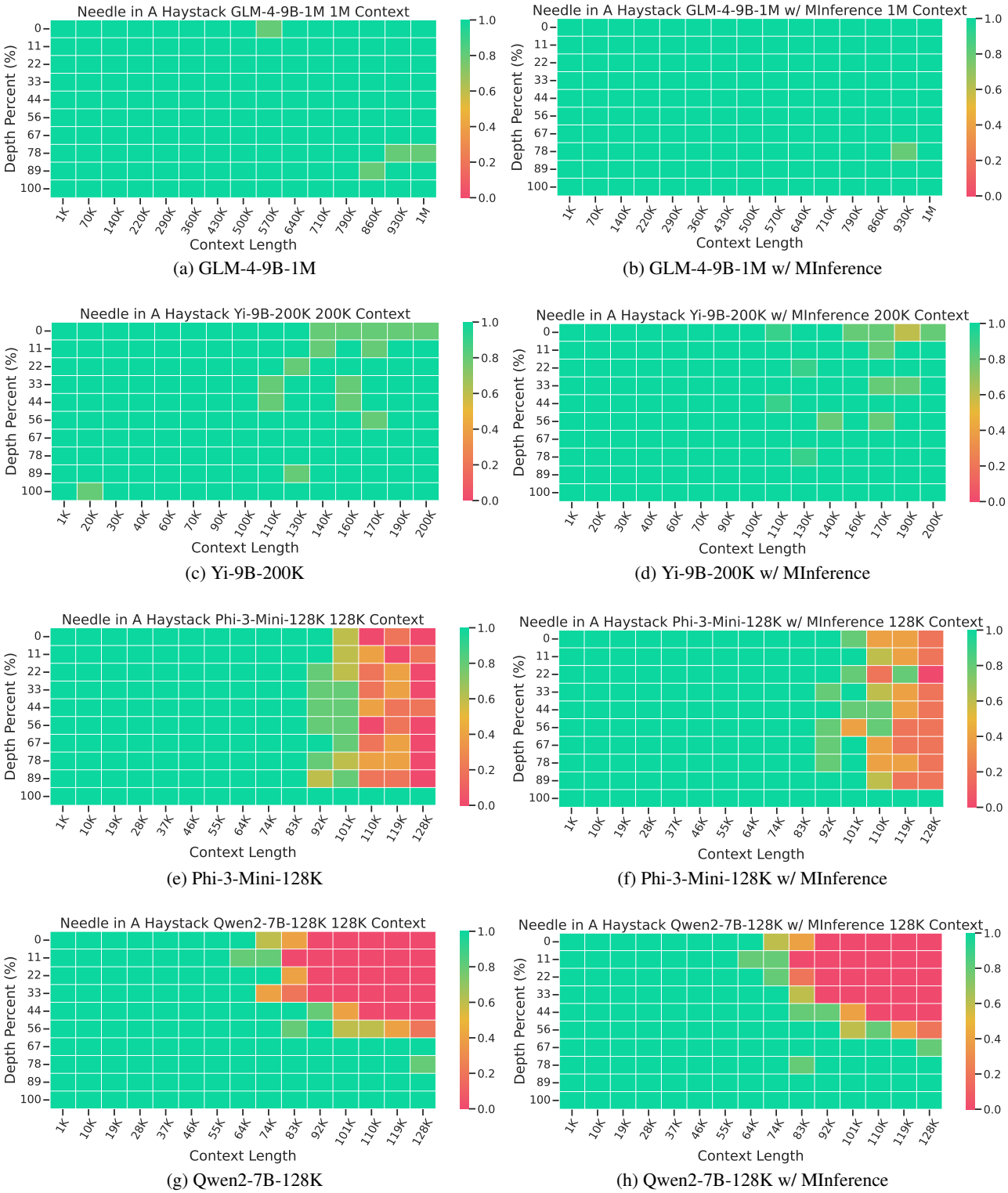
Similar vertical and slash line sparse patterns have been discovered in BERT (Shi et al., 2021) and multi-modal LLMs (Wan et al., 2024). Additionally, as shown in Fig. 11, we analyzed the distribution of attention patterns in T5 across different heads. It is evident that there are vertical and slash sparse patterns even in bidirectional attention.

Recent studies (Wan et al., 2024) have analyzed sparse attention patterns in multi-modal LLMs, revealing the presence of vertical and slash patterns in models like LLaVA (Liu et al., 2024a) and InternVL (Chen et al., 2024). Using MInference for pre-filling stage inference acceleration holds great promise.

I. Case Study

Table 9 presents a comparison of the generation performance for various methods on the EN.SUM task (200K input length) from InfiniteBench based on the LLaMA-3-8B-262K model. The original summary provides a comprehensive and coherent narrative, detailing the Bronwyn family's trip to the Kindergarten and touching on themes such as nostalgia, loss, and the passage of time. StreamingLLM's summary, although looks coherent, introduces elements that are not present in the original story, leading to serious factual errors. For example, it mentions a boat trip to a school for boys and specific details like fishermen, sandwiches, and a spot where men were drowned. These details deviate from the original story, which is about the Bronwyn family preparing for a trip to the Kindergarten. In addition, the summaries generated by StreamingLLM with dilated and strided techniques are largely incoherent, consisting primarily of repetitive and nonsensical characters, indicating a failure to produce meaningful content. In stark contrast, the summary generated by our proposed method offers a detailed and coherent narrative, comparable to the original, with a clear depiction of the story's main events and themes. This includes the preparation of the Bronwyn family for their trip, the characterization of family members and guests, and the exploration of deeper themes such as love, marriage, and the search for meaning. The results demonstrate the superiority of our proposed method in generating high-quality, human-like summaries over the baseline methods.

Table 10 compares the performance of various methods on the Retrieve.KV task (200K input length) using the LLaMA-3-8B-262K model. The original method demonstrates perfect retrieval, correctly predicting the exact strings of the ground truth for both examples. StreamingLLM, again, generates predictions that looks coherent and real, but factually incorrect. In addition, StreamingLLM with dilated and strided techniques, and our method with a static pattern, fail significantly, producing outputs that are either repetitive sequences of characters or nonsensical strings, indicating their inability to accurately retrieve the required key-value pairs. Our method, however, performs on par with the original, accurately retrieving and predicting the exact key-value pairs for both examples. This demonstrates the superior capability of our method in handling KV retrieval tasks, providing precise and reliable outputs consistent with the ground truth. The results highlight our method's effectiveness and robustness compared to the baselines, making it a reliable choice for such tasks.



(a) GLM-4-9B-1M (b) GLM-4-9B-1M w/ MInference

(c) Yi-9B-200K (d) Yi-9B-200K w/ MInference

(e) Phi-3-Mini-128K (f) Phi-3-Mini-128K w/ MInference

(g) Qwen2-7B-128K (h) Qwen2-7B-128K w/ MInference

Figure 12. Needle In A Haystack (Kamradt, 2023) results using GLM-4-9B-1M (GLM et al., 2024), Yi-9B-200K (Young et al., 2024), Phi-3-Mini-128K (Abdin et al., 2024), and Qwen2-7B-128K (Yang et al., 2024).

Algorithm 4 Vertical-Slash Index

Input: vertical indexes $\boldsymbol{i}_v \in \mathbb{N}^{k_v}$, slash indexes $\boldsymbol{i}_s \in \mathbb{N}^{k_s}$

 # Sort vertical and slash indexes
$\boldsymbol{i}_v \leftarrow \text{IncrementalSort}\,(\boldsymbol{i}_v)$
$\boldsymbol{i}_s \leftarrow \text{DescendingSort}\,(\boldsymbol{i}_s)$

 # Calculate block number (block_size B)
$N \leftarrow \lceil \frac{S}{B} \rceil$

 # Initialize outputs
block count $\boldsymbol{c}_{\text{blk}} \in \mathbb{N}^N$, block index $\boldsymbol{i}_{\text{blk}} \in \mathbb{N}^{N \times k_s}$, column count $\boldsymbol{c}_{\text{col}}\mathbb{N}^N$, column index $\boldsymbol{i}_{\text{col}} \in \mathbb{N}^{N \times k_v}$

 # Parallelized in GPU
for $i \leftarrow 1$ to N **do**
  $j_v \leftarrow 1$
  # Find the first slash line that crosses the row
  $j_s \leftarrow \text{biset\_left}(\boldsymbol{i}_s, i \times B)$
  # Define the range by slash index
  $r_{\text{start}} \leftarrow (i-1) \times B - \boldsymbol{i}_s^{j_s}$
  $r_{\text{end}} \leftarrow i \times B - \boldsymbol{i}_s^{j_s}$
  # Merge points (vertical indexes) and ranges (slash indexes)
  while $s_v \leq k_s$ **do**
    if $j_v \leq k_v$ **and** $\boldsymbol{i}_v^{j_v} < r_{\text{end}}$ **then**
      # Record the point if not in the range
      if $\boldsymbol{i}_v^{j_v} < r_{\text{start}}$ **then**
        $\boldsymbol{c}_{\text{col}}^i \leftarrow \boldsymbol{c}_{\text{col}}^i + 1, \boldsymbol{i}_{\text{col}}^{i,\boldsymbol{c}_{\text{col}}^i} \leftarrow \boldsymbol{i}_v^{j_v}$
      end
      $j_v \leftarrow j_v + 1$
    else
      $s_v \leftarrow s_v + 1$
      # Update the range
      if $(i-1) \times B - \boldsymbol{i}_s^{j_s} > r_{\text{end}}$ **then**
        # Record the last range
        $s \leftarrow r_{\text{start}}$
        while $s < r_{\text{end}}$ **do**
          $\boldsymbol{c}_{\text{blk}}^i \leftarrow \boldsymbol{c}_{\text{blk}}^i + 1$
          $\boldsymbol{i}_{\text{blk}}^{i,\boldsymbol{c}_{\text{blk}}^i} \leftarrow s$
          $s \leftarrow s + B$
        end while
        # Calculate the new range
        $r_{\text{start}} \leftarrow (i-1) \times B - \boldsymbol{i}_s^{j_s}$
        $r_{\text{end}} \leftarrow i \times B - \boldsymbol{i}_s^{j_s}$
      else
        # Extend the range
        $r_{\text{end}} \leftarrow r_{\text{end}} + B$
      end
    end
  end while
  # Record the last range
  $s \leftarrow r_{\text{start}}$
  while $s < r_{\text{end}}$ **do**
    $\boldsymbol{c}_{\text{blk}}^i \leftarrow \boldsymbol{c}_{\text{blk}}^i + 1$
    $\boldsymbol{i}_{\text{blk}}^{i,\boldsymbol{c}_{\text{blk}}^i} \leftarrow s, s \leftarrow s + B$
  end while
end for
return $\boldsymbol{c}_{\text{blk}}, \boldsymbol{i}_{\text{blk}}, \boldsymbol{c}_{\text{col}}, \boldsymbol{i}_{\text{col}}$

Algorithm 5 Vertical-Slash Flash Attention

Input: $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{S \times d_h}$, block count $\boldsymbol{c}_{\text{blk}} \in \mathbb{N}^N$, block index $\boldsymbol{i}_{\text{blk}} \in \mathbb{N}^{N \times k_s}$, column count $\boldsymbol{c}_{\text{col}} \in \mathbb{N}^N$, column index $\boldsymbol{i}_{\text{col}} \in \mathbb{N}^{N \times k_v}$
Scale $\tau \leftarrow \sqrt{\frac{1}{d_h}}$
Initialize $\boldsymbol{O} \leftarrow (0)^{S \times d_h} \in \mathbb{R}^{S \times d_h}$

 # Parallelized in GPU
for $i \leftarrow 1$ to N **do**
  Load $\boldsymbol{Q}_{\text{chip}} \leftarrow \boldsymbol{Q}^{i \times B:(i+1) \times B} \in \mathbb{R}^{B \times d_h}$
  Initialize $\boldsymbol{O}_{\text{chip}} \leftarrow (0)^{B \times d_h} \in \mathbb{R}^{B \times d_h}$
  Initialize $\boldsymbol{m} \leftarrow (-\inf)^B \in \mathbb{R}^B$
  Initialize $\boldsymbol{l} \leftarrow (0)^B \in \mathbb{R}^B$
  # Loop through block indexes: block sparse flash attention
  for $j \leftarrow 1$ to $\boldsymbol{c}_{\text{blk}}^i$ **do**
    Block start $s \leftarrow \boldsymbol{i}_{\text{blk}}^{i,j}$
    Load $\boldsymbol{K}_{\text{chip}} \leftarrow \boldsymbol{K}^{s:s+B} \in \mathbb{R}^{B \times d_h}$
    Load $\boldsymbol{V}_{\text{chip}} \leftarrow \boldsymbol{V}^{s:s+B} \in \mathbb{R}^{B \times d_h}$
    $\boldsymbol{S} \leftarrow \tau \boldsymbol{Q}_{\text{chip}} \boldsymbol{K}_{\text{chip}}^T$
    $\boldsymbol{S} \leftarrow \text{mask}(\boldsymbol{S})$
    $\boldsymbol{m}_{new}^i \leftarrow \max(\boldsymbol{m}^i, \text{rowmax}(\boldsymbol{S})) \in \mathbb{R}^B$
    $\boldsymbol{S} \leftarrow \boldsymbol{S} - \boldsymbol{m}_{new}^i$
    $\boldsymbol{P} \leftarrow \exp(\boldsymbol{S})$
    $\boldsymbol{l}_{new}^i \leftarrow \text{rowsum}(\boldsymbol{S}))$
    $\boldsymbol{\alpha} \leftarrow \exp(\boldsymbol{m}^i - \boldsymbol{m}_{new}^i)$
    $\boldsymbol{l}^i \leftarrow \boldsymbol{\alpha} \boldsymbol{l}^i + \boldsymbol{l}_{new}^i$
    $\boldsymbol{O}_{\text{chip}} \leftarrow \boldsymbol{\alpha} \boldsymbol{O}_{\text{chip}} + \boldsymbol{P} \boldsymbol{V}_{\text{chip}}$
  end for
  # Loop through column indexes : PIT sparse flash attention
  $j \leftarrow 0$
  while $j < \boldsymbol{c}_{\text{col}}^j$ **do**
    $\boldsymbol{cols} \leftarrow \boldsymbol{i}_{\text{col}}^{i,j:j+B} \in \mathbb{N}^B$
    Load $\boldsymbol{K}_{\text{chip}} \leftarrow \boldsymbol{K}^{\boldsymbol{cols}} \in \mathbb{R}^{B \times d_h}$
    Load $\boldsymbol{V}_{\text{chip}} \leftarrow \boldsymbol{V}^{\boldsymbol{cols}} \in \mathbb{R}^{B \times d_h}$
    $\boldsymbol{S} \leftarrow \tau \boldsymbol{Q}_{\text{chip}} \boldsymbol{K}_{\text{chip}}^T$
    $\boldsymbol{S} \leftarrow \text{mask}(\boldsymbol{S})$
    $\boldsymbol{m}_{new}^i \leftarrow \max(\boldsymbol{m}^i, \text{rowmax}(\boldsymbol{S})) \in \mathbb{R}^B$
    $\boldsymbol{S} \leftarrow \boldsymbol{S} - \boldsymbol{m}_{new}^i$
    $\boldsymbol{P} \leftarrow \exp(\boldsymbol{S})$
    $\boldsymbol{l}_{new}^i \leftarrow \text{rowsum}(\boldsymbol{S}))$
    $\boldsymbol{\alpha} \leftarrow \exp(\boldsymbol{m}^i - \boldsymbol{m}_{new}^i)$
    $\boldsymbol{l}^i \leftarrow \boldsymbol{\alpha} \boldsymbol{l}^i + \boldsymbol{l}_{new}^i$
    $\boldsymbol{O}_{\text{chip}} \leftarrow \boldsymbol{\alpha} \boldsymbol{O}_{\text{chip}} + \boldsymbol{P} \boldsymbol{V}_{\text{chip}}$
    $j \leftarrow j + B$
  end while
  # Write outputs
  $\boldsymbol{O}_{\text{chip}} \leftarrow \text{diag}(\boldsymbol{l}^i)^{-1} \boldsymbol{O}_{\text{chip}}$
  Save $\boldsymbol{O}_i \leftarrow \boldsymbol{O}_{\text{chip}}$
end for

Table 9. Comparison of generated results by different methods using LLaMA-3-8B-Instruct-262K in summarization task (Zhang et al., 2024a).

Methods	Summary
Original	The story follows the Bronwyn family and their guests as they prepare for a day trip to the Kindergarten, a mysterious destination that holds great significance for them. The story is told from multiple perspectives, including that of Pete Lucy, who is staying with the family and is struggling to find her place in their lives. As the day progresses, the family members prepare for their trip, and Pete reflects on her own past experiences with the Bronwyns. The story ends with the family setting off on their journey, leaving Pete alone on the lawn to contemplate her own feelings and the meaning of life. The Kindergarten is a symbol of the family's past and their memories, and the story explores themes of nostalgia, loss, and the passage of time. Ultimately, the story suggests that even though the past cannot be changed, it can still be celebrated and remembered.
StreamingLLM	The story follows a family on a boat trip to the Kindergarten, a school for boys. The father, Mr Bronwyn, is reading a book and seems to be in a trance. The children, Seb and Annalisa, are excited about the trip but also a little scared. They pass by a group of fishermen and Mr Bronwyn shares out sandwiches. They come across a spot where three men were drowned and Mr Bronwyn takes out his watch, making a calculation. They reach the Kindergarten and Mr Bronwyn jumps out of the boat, holding a parcel. The story ends with Pete Lucy, who has been watching the family from the shore, feeling relieved that the trip is over and that Mr Bronwyn has reached his destination. The story is told from multiple perspectives, including Pete Lucy's, who is watching the family from the shore and reflecting on his own life and relationships.
StreamingLLM w/ dilated	A (
StreamingLLM w/ strided	The GraphicsUnit36.OOAKA new (
Ours w/ static	def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def
Ours	The story follows the Bronwyn family and their guests as they prepare for a trip to the Kindergarten, a remote island off the coast of Scotland. The family is made up of Mr. Bronwyn, a philosopher, and his wife, Mrs. Bronwyn, who is known for her beauty and her ability to surround and protect her family. The guests include Kevin Emma, a young man who is studying for a fellowship, and Castor Kiki, a botanist who is also a widower. The story explores themes of love, marriage, and the search for meaning in life. It also touches on the idea of the other, or the person who is different from oneself and cannot be understood or judged by conventional standards. Ultimately, the story is about the search for truth and the desire to connect with others on a deeper level.

Table 10. Comparison of generated results by different methods using LLaMA-3-8B-Instruct-262K in KV Retrieval task (Zhang et al., 2024a).

Methods	Summary
Original	1. Ground Truth: 5e6b7b90-710d-4953-9b18-3e96b2cadbf2 Prediction: "5e6b7b90-710d-4953-9b18-3e96b2cadbf2" 2. Ground Truth: f1ec3dd8-b6e7-4af5-a589-fda0b7693f93 Prediction: "f1ec3dd8-b6e7-4af5-a589-fda0b7693f93" (a string of length 32).
StreamingLLM	1. Ground Truth: 5e6b7b90-710d-4953-9b18-3e96b2cadbf2 Prediction: 2b114db0-d87e-42d2-9b4c-0b1f115976ad.assistant 2. Ground Truth: bbf3fa73-6217-4e6e-923c-8349bd286c3d Prediction: "d829ce18-6339-4f77-8c04-31fc7ec33619".
StreamingLLM w/ dilated	1. Ground Truth: 5e6b7b90-710d-4953-9b18-3e96b2cadbf2 Prediction: 5 (2. Ground Truth: f1ec3dd8-b6e7-4af5-a589-fda0b7693f93 Prediction: "def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 def solverome2
StreamingLLM w/ strided	1. Ground Truth: 5e6b7b90-710d-4953-9b18-3e96b2cadbf2 Prediction: "def solverome2 def solverome2 def solverome2 def solverome2 def solverome2 2. Ground Truth: f1ec3dd8-b6e7-4af5-a589-fda0b7693f93 Prediction: "0 (
Ours w/ static	1. Ground Truth: 5e6b7b90-710d-4953-9b18-3e96b2cadbf2 Prediction: "def solverome2 def solverome2 def solverome2 def solverome2 def 2. Ground Truth: f1ec3dd8-b6e7-4af5-a589-fda0b7693f93 Prediction: "def solverome2 def solverome2 def solverome2 def solverome2 def
Ours	1. Ground Truth: 5e6b7b90-710d-4953-9b18-3e96b2cadbf2 Prediction: "5e6b7b90-710d-4953-9b18-3e96b2cadbf2" 2. Ground Truth: f1ec3dd8-b6e7-4af5-a589-fda0b7693f93 Prediction: "f1ec3dd8-b6e7-4af5-a589-fda0b7693f93" (a string of length 32).