ADAMAS: HADAMARD SPARSE ATTENTION FOR EFFICIENT LONG-CONTEXT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) now support context windows of hundreds of thousands to millions of tokens, enabling applications such as long-document summarization, large-scale code synthesis, multi-document question answering and persistent multi-turn dialogue. However, such extended contexts exacerbate the quadratic cost of self-attention, leading to severe latency in autoregressive decoding. Existing sparse attention methods reduce these costs by attending to only a subset of key-value (KV) tokens, but fixed or coarse-grained selection strategies can miss important tokens or retain substantial redundancy under constrained token budgets. We introduce **Adamas**, an accurate and efficient token-level sparse attention mechanism for long-context inference. Adamas applies the Hadamard transform, bucketization and 2-bit compression to produce compact representations, and uses lightweight distance-based estimation for efficient top-k candidate selection. The selected positions are then used to gather the corresponding original KV states for sparse attention. Comprehensive evaluations show that Adamas maintains near-lossless performance within the effective context length of Llama-3.1-8B-Instruct. In efficiency evaluations, Adamas achieves up to $6.5\times$ self-attention speedup and $1.7\times$ end-to-end decoding speedup at a 32K context length. Adamas also maintains performance comparable to full attention on challenging long-form reasoning benchmarks, including AIME 2024 and AIME 2025.

1 INTRODUCTION

The rapid progress of LLMs has dramatically extended their affordable context windows. Contemporary systems such as Anthropic's Claude Sonnet 4 support up to 1M tokens (Anthropic, 2025), while OpenAI's GPT-5 effectively handles contexts up to 400K tokens (OpenAI, 2025). These expanded capacities enable advanced applications, including long-document summarization (Chang et al., 2024), large-scale code synthesis (Nijkamp et al., 2023; Dainese et al., 2024), multi-document question answering (Wu et al., 2025a; Wang et al., 2024), and persistent multi-turn dialogue (Wu et al., 2025b). By processing extensive information without manual segmentation, LLMs are increasingly able to tackle tasks that demand global coherence and long-term memory. However, the quadratic complexity of self-attention (Zaheer et al., 2020; Vaswani et al., 2017), together with the growing KV cache, introduces substantial per-token latency and memory overhead during inference.

Sparse attention offers a promising solution by restricting each query to attend only a carefully selected subset of tokens (Yuan et al., 2025). This reduces the number of KV pairs involved in attention computation, lowering both computational complexity and memory access costs while largely preserving modeling capacity. Nevertheless, existing approaches often fall into two categories with notable limitations. Static sparsity patterns (e.g., StreamingLLM (Xiao et al., 2023)) are often hand-designed from empirical attention heatmaps, yielding fixed patterns, such as fixed local windows or vertical stripes. However, these patterns fail to capture the dynamic nature of query–key interactions, leading to low recall and degraded accuracy (See Table 1). In contrast, dynamic methods like Quest (Tang et al., 2024) adapt token selection during inference, but their page-level granularity remains overly coarse. This coarse selection introduces token redundancy and limits the achievable sparsity ratio, since higher sparsity levels lead to accuracy degradation.

In this paper, we propose **Adamas**, an accurate and efficient token-level sparse attention mechanism for long-context inference. Adamas uses compact query and key representations to efficiently

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

identify candidate KV tokens for each decoding query. Specifically, it first applies the Hadamard transform to queries and keys, which redistributes large-magnitude outliers and produces smoother feature distributions for coarse discretization. The transformed representations are then bucketized into compact 2-bit codes using calibrated thresholds and stored alongside the original KV cache with small additional memory overhead. During decoding, Adamas evaluates a lightweight distance-based score over the compressed representations to select the top-k candidate KV positions. The corresponding original KV states are then gathered for sparse attention.

This token-wise selection allows Adamas to reduce the number of KV states accessed and processed for each decoding query while preserving high-quality attention candidates. Comprehensive evaluations show that Adamas maintains near-lossless performance within the effective context length of Llama-3.1-8B-Instruct (Hsieh et al., 2024). In efficiency evaluations, Adamas achieves up to $6.5\times$ self-attention speedup and $1.7\times$ end-to-end decoding speedup at a 32K context length. Adamas also remains effective on long-form reasoning tasks, achieving performance comparable to full attention on both AIME 2024 (Zhang & Math-AI, 2024) and AIME 2025 (Zhang & Math-AI, 2025) with Qwen3-8B (Qwen Team, 2025) in reasoning mode. Our main contributions are as follows:

- We propose **Adamas**, a token-level sparse attention mechanism that combines Hadamard transform, bucketization, 2-bit compression and lightweight distance-based candidate selection to efficiently identify relevant KV pairs during decoding.
- We develop high-performance GPU kernels for Adamas, enabling up to $6.5\times$ self-attention speedup and $1.7\times$ end-to-end decoding speedup over full attention in long-context decoding.
- We conduct comprehensive evaluations across long-context understanding, retrieval, language modeling, and long-form reasoning, together with analyses of candidate-selection quality and the key design choices of Adamas.

2 PRELIMINARIES

In this section, we briefly review sparse attention and the Hadamard transform to provide background and facilitate a clearer understanding of our proposed method.

Sparse attention is a variant of the self-attention mechanism. The standard attention computation is defined as

$$O = \mathrm{Softmax}(\frac{QK^\top}{\sqrt{d}})V, \tag{1}$$

where Q, K, V, and O denote the query, key, value, and attention output, respectively. Since $QK^\top$ computes the query–key similarity, achieving high recall requires identifying the most relevant keys for each query. However, evaluating the similarity between all query–key pairs incurs quadratic complexity with respect to the sequence length, making it impractical for long contexts.

Sparse attention reduces this cost by restricting each query to attend to only a selected subset of keys. In general, let s_i denote an estimated relevance score between a query and the i-th key, where $i = 1, \ldots, N$ and N is the number of tokens. The selected index set is obtained by

$$\mathcal{I} = \mathrm{Topk}(s), \tag{2}$$

where $\mathrm{Topk}(\cdot)$ returns the indices of the k most relevant keys according to the scoring function. The corresponding sparse key and value tensors are then gathered as

$$K_{\mathcal{I}} = K^{\mathrm{all}}[\mathcal{I}], \qquad V_{\mathcal{I}} = V^{\mathrm{all}}[\mathcal{I}]. \tag{3}$$

Sparse attention is finally computed only over the selected tokens:

$$O = \mathrm{Softmax}\left(\frac{QK_{\mathcal{I}}^\top}{\sqrt{d}}\right) V_{\mathcal{I}}. \tag{4}$$

Hadamard transform, also known as the Walsh–Hadamard transform, is an orthogonal linear transform that performs a change of basis using Walsh functions. It corresponds to multiplication by a Hadamard matrix $\mathbf{H}_d \in \mathbb{R}^{d\times d}$, where $d = 2^r$. The Hadamard matrix has entries in $\{+1, -1\}$, and the normalized Hadamard transform is obtained by scaling $\mathbf{H}_d$ by $1/\sqrt{d}$. The smallest normalized Hadamard matrix is given by:

$$\mathbf{H}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5}$$

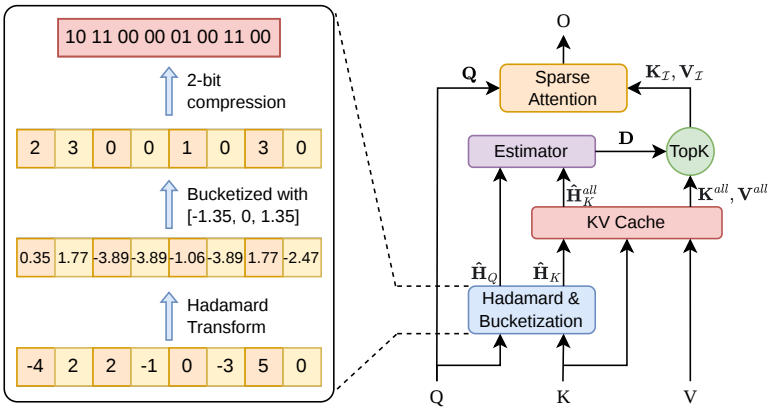


Figure 1: **Overview of Adamas.** Queries Q and keys K are processed by the Hadamard transform, followed by bucketization and 2-bit compression. The compressed query $\widehat{\mathbf{H}}_Q$ and keys $\widehat{\mathbf{H}}_K$ are compared using a lightweight distance-based estimator to select the top-k candidate positions. The corresponding original keys and values, $K_\mathcal{I}$ and $V_\mathcal{I}$, are then gathered for sparse attention with the original query Q. All boxed components are implemented as dedicated GPU kernels for efficiency.

Higher-order matrices are constructed recursively via the Kronecker product $\mathbf{H}_{2^r} = \mathbf{H}_2 \otimes \mathbf{H}_{2^{r-1}}$. This recursive structure enables an efficient Hadamard transform algorithm that computes the matrix–vector product $\mathbf{H}x$ in $\mathcal{O}(d \log_2 d)$ rather than the naive $\mathcal{O}(d^2)$. For dimensions d that are not exact powers of two, the existence of a Hadamard matrix is not guaranteed. In such cases, one may exploit factorizations $d = 2^r m$, where m is the order of a known Hadamard matrix, and apply the Kronecker construction $\mathbf{H}_d = \mathbf{H}_{2^r} \otimes \mathbf{H}_m$, yielding a transform with complexity $\mathcal{O}(d(m + r))$.

3 Method

In this section, we present Adamas, an efficient variant of the Transformer sparse attention mechanism that could substantially reduce computation overhead while preserving model accuracy. We show the workflow of Adamas in Figure 1.

3.1 Design Rationale

Adamas is designed to construct a compact representation for efficient token-level candidate selection during decoding. We first apply an orthogonal Hadamard transform to queries and keys before discretization. Since $\mathbf{H}\mathbf{H}^\top = \mathbf{I}$, applying the same transform to both preserves their dot products:

$$(Q\mathbf{H})(K\mathbf{H})^\top = Q(\mathbf{H}\mathbf{H}^\top)K^\top = QK^\top \tag{6}$$

Thus, the Hadamard transform changes the representation basis without altering the query–key dot-product similarity before discretization. Directly discretizing query and key vectors, however, can be sensitive to large-magnitude outliers. The Hadamard transform redistributes feature magnitudes more evenly across dimensions and suppresses extreme values, yielding smoother representations that are better suited for coarse quantization (Elhage et al., 2023; Ashkboos et al., 2024). We therefore bucketize the transformed queries and keys into compact 2-bit codes and use them for lightweight candidate estimation. The structured sign pattern of the Hadamard transform also enables an efficient implementation via the fast Hadamard transform (Dao, 2024; IBM & Meta, 2024). The effects of the Hadamard transform, compression granularity and candidate selection are evaluated in Sec. 4.4 and Appendix B.

3.2 Adamas

As shown in Algorithm 1, Adamas augments the standard Transformer attention pipeline with three key innovations: 1) applying the Hadamard transform to queries and keys, yielding what we refer

Algorithm 1 Workflow of Adamas

Input: Query $\mathbf{Q}$, Key $\mathbf{K}$, and Value $\mathbf{V} \in \mathbb{R}^d$, Hadamard matrix $\mathbf{H} \in \mathbb{R}^{d \times d}$
Output: Attention output matrix $\mathbf{O}$

$\mathbf{H}_Q \leftarrow \mathbf{Q}\mathbf{H}$; $\mathbf{H}_K \leftarrow \mathbf{K}\mathbf{H}$ ▷ Apply Hadamard transform
$\widehat{\mathbf{H}}_Q, \widehat{\mathbf{H}}_K \leftarrow \text{BucketizeCompress}(\mathbf{H}_Q, \mathbf{H}_K)$ ▷ Bucketization and compression
$\widehat{\mathbf{H}}_K^{\text{all}}, \mathbf{K}^{\text{all}}, \mathbf{V}^{\text{all}} \leftarrow \text{KVCache.update}(\widehat{\mathbf{H}}_K, \mathbf{K}, \mathbf{V})$
$D \leftarrow \text{DistanceEstimator}(\widehat{\mathbf{H}}_Q, \widehat{\mathbf{H}}_K^{\text{all}})$ ▷ Estimate query–key relevance
$I \leftarrow \text{Topk}_{\min}(D)$ ▷ Select top-k candidate indices
$\mathbf{K}_\mathcal{I}, \mathbf{V}_\mathcal{I} \leftarrow \mathbf{K}^{\text{all}}[I], \mathbf{V}^{\text{all}}[I]$ ▷ Gather selected KV states
$\mathbf{O} \leftarrow \text{SparseAttention}(\mathbf{Q}, \mathbf{K}_\mathcal{I}, \mathbf{V}_\mathcal{I})$
return $\mathbf{O}$

to as Hadamard vectors; 2) bucketization of the Hadamard vectors; and 3) a lightweight distance-based estimation over the compressed representations. These components work jointly to enable faster attention computation with modest memory overhead and near-lossless accuracy.

Hadamard transform applied to queries and keys. Formally, given queries and keys $Q, K \in \mathbb{R}^d$, and a Hadamard matrix $\mathbf{H} \in \mathbb{R}^{d \times d}$, we compute their Hadamard-transformed representations as

$$H_Q = Q\mathbf{H}, \qquad H_K = K\mathbf{H}, \tag{7}$$

Bucketization. For efficient computation and storage, each element in $H_Q, H_K \in \mathbb{R}^d$ is discretized into one of four levels using predefined thresholds $\{B_1, B_2, B_3\}$, yielding the bucketized Hadamard vectors $\widehat{\mathbf{H}}_Q$ and $\widehat{\mathbf{H}}_K$. The bucketization operator $B(\cdot)$ is defined as

$$B(x) = \Sigma_{i=1}^{3} \mathbb{I}(x > B_i), \tag{8}$$

where $\mathbb{I}(\cdot)$ denotes the indicator function.

We determine the bucketization scale using a lightweight offline calibration. On a 10% subsample of representative LongBench datasets, the elements of both H_Q and H_K are approximately zero-centered, with most values concentrated around the origin. We therefore use their empirical standard deviations $\sigma_X, X \in \{Q, K\}$ as scale parameters and set the three thresholds to $\{-\sigma_X, 0, \sigma_X\}$. The resulting calibrated values are $\sigma_Q = 1.35, \sigma_K = 2.26$, which are fixed during inference. Appendix A evaluates the sensitivity of this choice.

Distance-based candidate estimation. After bucketization, each element of $\widehat{\mathbf{H}}_Q$ and $\widehat{\mathbf{H}}_K$ is represented by a 2-bit code, and every eight codes are packed into a single 16-bit value. The compressed $\widehat{\mathbf{H}}_K$ is stored alongside the original KV cache, incurring only a small additional memory overhead. We estimate the relevance of each key using a lightweight distance function over the compressed representations. In our default implementation, we use the ℓ_1 distance:

$$\mathbf{D}(\widehat{\mathbf{H}}_Q, \widehat{\mathbf{H}}_K) = \left\| \widehat{\mathbf{H}}_Q - \widehat{\mathbf{H}}_K \right\|_1. \tag{9}$$

A smaller distance indicates a higher-priority candidate. Since both $\widehat{\mathbf{H}}_Q$ and $\widehat{\mathbf{H}}_K$ are represented by low-bit integers, the distance can be evaluated using lightweight integer operations over the packed representations, avoiding floating-point similarity computation during candidate selection. The compressed representations and distance estimator are used only to construct a candidate set. The corresponding original key and value states are then gathered for sparse attention. We evaluate the quality of the selected candidate sets in Appendix B and the effect of distance metric in Sec. 4.4.

Top-k selection with compressed representations. Following the sparse attention formulation in Eq. 2–4, our method estimates token relevance using distances in the compressed representation space. Given the compressed query $\widehat{\mathbf{H}}_Q$ and all compressed keys $\widehat{\mathbf{H}}_K^{\text{all}}$, we compute a distance score D_i for each cached token and select the k positions with the smallest distances:

$$\mathcal{I} = \text{TopK}_{\min}(\mathbf{D}). \tag{10}$$

The corresponding original key and value states are then gathered as $\mathbf{K}_\mathcal{I} = \mathbf{K}^{\text{all}}[\mathcal{I}]$ and $\mathbf{V}_\mathcal{I} = \mathbf{V}^{\text{all}}[\mathcal{I}]$ for sparse attention.

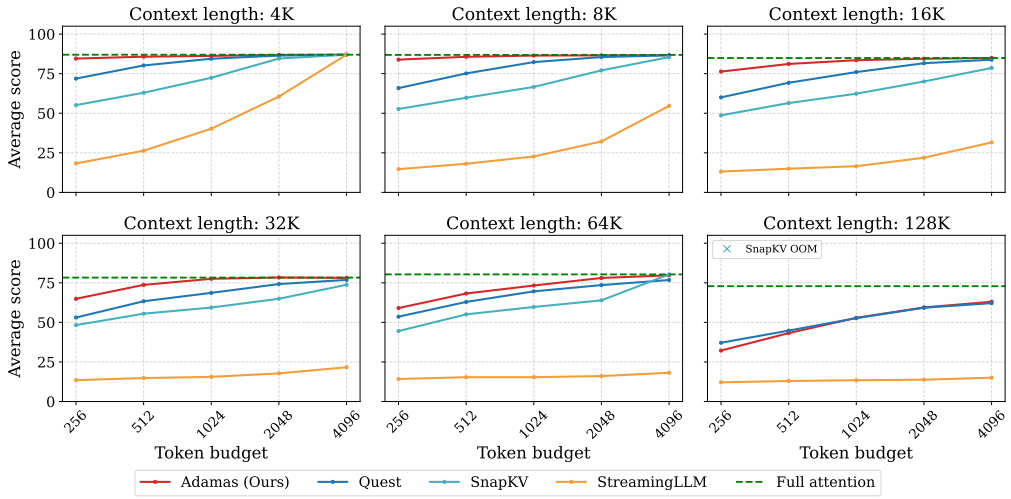


Figure 2: **Evaluation results on RULER.** Adamas exhibits the smallest performance drop compared to full attention while maintaining high sparsity.

Group-wise selection for grouped-query attention (GQA). For GQA models, we exploit the fact that multiple query heads share the same key-value head. Let $\mathcal{G}_g$ denote the set of query heads associated with the g-th key-value head. For each query head $q \in \mathcal{G}_g$, let $D_q \in \mathbb{R}^N$ be its distance scores to the N tokens. Instead of performing Top-k selection independently for each query head, we compute a group-level distance score by taking the element-wise minimum over all query heads in the same GQA group:

$$D_i^{(g)} = \min_{q \in \mathcal{G}_g} D_{q,i}, \qquad i = 1, \ldots, N. \tag{11}$$

We then select the k positions with the smallest group-level distances $I^{(g)} = \mathrm{Topk}_{\min}\left(D^{(g)}\right)$. The selected indices are shared by all query heads in the same GQA group $I_q = I^{(g)}, \forall q \in \mathcal{G}_g$. This group-wise selection reduces memory accesses and improves computational efficiency, while maintaining comparable retrieval quality.

Customized CUDA Kernels. To fully exploit the compact representation and bit-wise computation enabled by Adamas, we implement a set of specialized high-performance CUDA kernels that optimize each stage of the attention pipeline. Specifically, we design three customized kernels corresponding to fused Hadamard transform + bucketization + compression, fused distance-based candidate estimation + top-k selection, and sparse attention. In our default ℓ_1 implementation, the distance estimation is efficiently realized with lightweight integer operations over packed 2-bit codes.

4 Experiments

In this section, we evaluate Adamas in terms of both efficacy and efficiency. We also conduct ablation studies on the Hadamard transform, bucketization, and distance metrics to better understand the contribution of each component. Code is publicly available at `https://anonymous.4open.science/r/Adamas-36EA`.

4.1 Setup

Models. We use Llama-3.1-8B-Instruct (Meta, 2024) and Qwen3-8B (Qwen Team, 2025) as representative GQA models, LongChat-7b-v1.5-32k (Li et al., 2023) and Yarn-Llama-2-7b-128k (Peng et al., 2024) as representative MHA models.

Tasks. We consider four categories of evaluation benchmarks. First, we evaluate long-context understanding using RULER (Hsieh et al., 2024) and LongBench (Bai et al., 2024). RULER provides synthetic stress tests for long-context capabilities, whereas LongBench covers diverse real-world

Table 1: Performance comparison on LongBench across models with different attention mechanisms (GQA and MHA). Scores denote normalized score (%) over budgets ranging from 256 to 4096, normalized by full-attention performance. The best results are shown in **bold**.

Method	Llama-3.1-8B-Instruct (GQA)	Qwen3-8B (GQA)	LongChat-7b-v1.5-32k (MHA)	Average
Quest	86.96	76.92	94.97	86.28
SnapKV	84.86	83.64	86.44	84.98
StreamingLLM	82.32	81.22	48.61	70.72
Adamas (Ours)	**97.65**	**98.04**	**100.65**	**98.78**

tasks: single-document QA (NarrativeQA (Kočiskỳ et al., 2018), Qasper (Dasigi et al., 2021), MultiFieldQA (Bai et al., 2024)), multi-document QA (HotpotQA (Yang et al., 2018)), summarization (GovReport (Huang et al., 2021)), and few-shot learning (TriviaQA (Joshi et al., 2017)). Second, we evaluate long-form mathematical reasoning on AIME 2024 and AIME 2025 using Qwen3-8B in reasoning mode. We use greedy decoding and repeat each evaluation three times, obtaining consistent results across runs. These benchmarks allow us to examine whether sparse attention remains effective over long reasoning trajectories. Third, we evaluate on the passkey retrieval task (Peng et al., 2024), which measures retrieval capability in long contexts. Fourth, we use PG19 (Rae et al., 2019), a long-form language modeling dataset, to assess long-context language-modeling quality.

Baselines. For fair comparison, we benchmark against training-free sparse attention methods: StreamingLLM (Xiao et al., 2023) for static-sparse-mask method, and SnapKV (Li et al., 2024) and Quest (Tang et al., 2024) for dynamic-KV-selection methods.

4.2 Efficacy evaluation

RULER. We evaluate Llama-3.1-8B-Instruct on all RULER tasks and report the average score. As shown in Figure 2, Adamas consistently outperforms other baselines across context lengths and token budgets. Within the 32k effective context length of Llama-3.1-8B-Instruct ((Hsieh et al., 2024)), Adamas achieves near-lossless performance, closely matching full attention scores. In comparison, StreamingLLM consistently underperforms full attention because its KV cache eviction strategy discards many task-relevant tokens. SnapKV shows mediocre accuracy under small and moderate budgets and only becomes competitive at a 4096-token budget, while Quest performs well when the token budget exceeds 1024 but degrades sharply under tighter budgets. These results indicate that Adamas can more reliably identify and preserve task-relevant KV pairs across context lengths.

LongBench. Table 1 reports the LongBench results across models with different attention mechanisms, where scores are normalized by full-attention performance. Overall, Adamas achieves the best average performance, reaching 98.78%, which substantially outperforms Quest (86.28%), SnapKV (84.98%), and StreamingLLM (70.72%). These results show that Adamas preserves nearly full-attention performance not only on GQA models but also on MHA models, demonstrating its robustness across attention architectures. Detailed results are provided in Appendix F.

Long-reasoning evaluations. We evaluate Adamas on both AIME 2024 and the more recent AIME 2025 benchmark using Qwen3-8B in reasoning mode, with `max_new_tokens` = 38,912 to accommodate long reasoning traces. As shown in Table 2, Adamas remains competitive with full attention on both benchmarks. On AIME 2024, it obtains 23/30 with a 1024-token budget and 20/30 with a 4096-token budget, compared with 20/30 under full attention. On AIME 2025, Adamas scores 17/30, 19/30, and 21/30 with budgets of 1024, 2048, and 4096, respectively, compared with 19/30 under full attention. These results show that Adamas preserves long-form reasoning performance even when attention is restricted to a small subset of the KV cache.

Table 2: **Long-reasoning evaluation on AIME 2024 and AIME 2025 with Qwen3-8B in reasoning mode.** Scores denote the number of correctly solved problems out of 30.

Benchmark	1024	2048	4096	Full Attn.
AIME 2024	**23**	21	20	20
AIME 2025	17	19	**21**	19

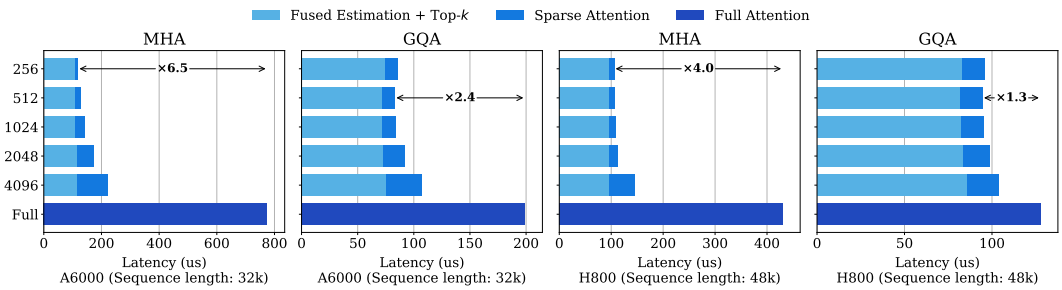


Figure 3: Kernel-level breakdown of Adamas self-attention compared to full attention (FlashInfer).

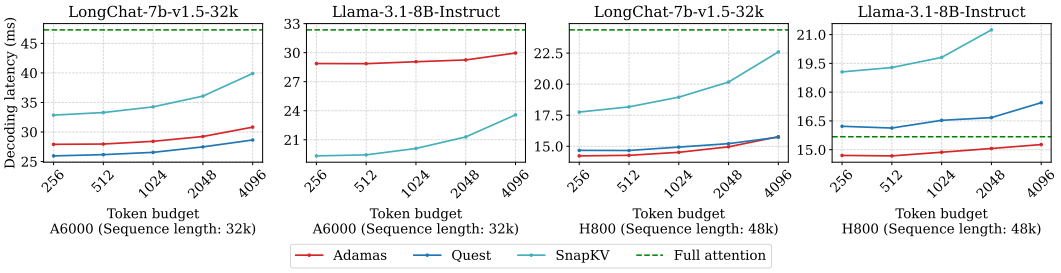


Figure 4: End-to-end decoding latency (ms) comparison. Out-of-Memory (OOM) results omitted.

Passkey results. We evaluate passkey retrieval on Llama-3.1-8B-Instruct, Qwen3-8B and Yarn-Llama-2-7b-128k with 100k-token inputs. As shown in Table 5, Adamas consistently achieves the best or comparable performance across most token budgets, with particularly large gains in low-budget regimes. Detailed comparisons and analysis are provided in Appendix C.

Perplexity results. We evaluate perplexity on the PG19 dataset using Llama-3.1-8B-Instruct, Qwen3-8B, and LongChat-7b-v1.5-32k with a sequence length of 32K. Across all models, Adamas consistently achieves perplexity close to full attention, demonstrating strong accuracy and robustness under long-context settings. Detailed results are provided in Appendix D.

4.3 Efficiency Evaluation

We conduct efficiency evaluations with Llama-3.1-8B-Instruct (GQA) and LongChat-7b-v1.5-32k (MHA) on NVIDIA RTX A6000 and NVIDIA H800.

Kernels. As shown in Figure 3, Adamas achieves consistent acceleration across different token budgets and sequence lengths. Specifically, Adamas delivers up to $6.5\times$ speedup over full attention with MHA and up to $2.4\times$ speedup with GQA, both implemented using FlashInfer (Ye et al., 2025), while maintaining near-lossless accuracy. Since GQA reduces the number of key-value heads, it already provides substantial acceleration compared with MHA; nevertheless, Adamas still achieves additional speedup under GQA, demonstrating its robustness across different attention architectures. The fused Hadamard transform and bucketization-compression introduce negligible overhead and are therefore omitted from further analysis.

End-to-end. Figure 4 shows that Adamas provides stable end-to-end decoding acceleration across all token budgets, GPU platforms, and model architectures. As the token budget increases from 256 to 4096, Adamas maintains consistently low latency and achieves up to $1.8\times$ speedup on MHA models and $1.1\times$ speedup on GQA models. The gain on GQA is particularly notable: since GQA already reduces the number of key-value heads and therefore substantially lowers the attention cost compared with MHA, the additional speedup achieved by Adamas demonstrates its robustness and practical acceleration capability under more efficient attention architectures.

In comparison, on A6000 with Llama-3.1-8B-Instruct at a 32K context length, Quest encounters OOM and therefore cannot complete the decoding evaluation. SnapKV provides only limited acceleration in most settings. On H800 with Llama-3.1-8B-Instruct at a 48K context length, both Quest

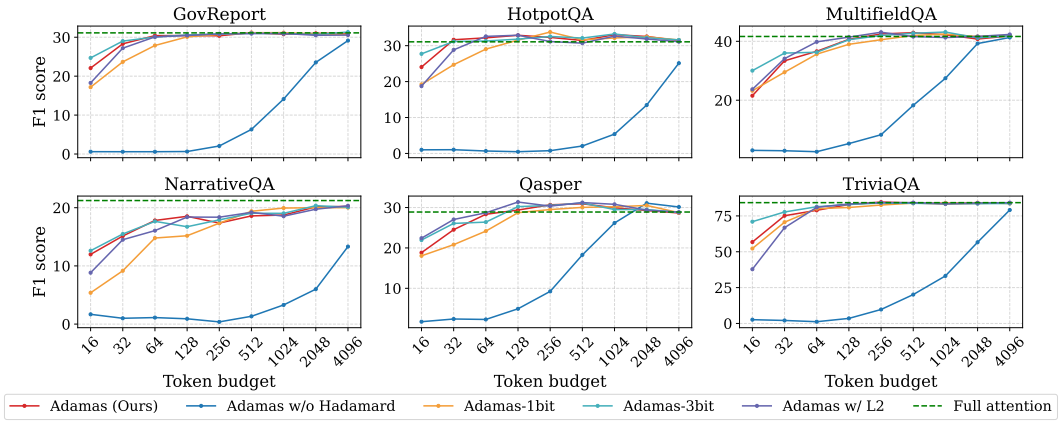


Figure 5: Ablation studies on the Hadamard transform, bucketization, and distance metrics.

and SnapKV are slower than full attention. For Quest, this is partly due to its lack of native GQA support: its current implementation converts the shared GQA key-value heads into an MHA-style representation before candidate selection and sparse attention, introducing additional memory and computation overhead that becomes non-negligible at long context lengths. In contrast, Adamas performs group-wise candidate selection directly over the native GQA structure without expanding the KV heads to MHA. At 56K, both Quest and SnapKV encounter out-of-memory failures on Llama-3.1-8B-Instruct, whereas Adamas completes the evaluation across all token budgets. Detailed latency results are provided in Appendix G.

4.4 Ablation Study

In this part, we conduct ablation studies on the Hadamard transform, bucketization, and distance metrics. We benchmark three variants of Adamas on LongBench using LongChat-7b-v1.5-32k (other models show similar results): (1) Adamas without the Hadamard transform, (2) Adamas with different bucketization granularities, and (3) Adamas with ℓ_2 distance (replacing the default ℓ_1 distance). The results are shown in Figure 5, with detailed data provided in Appendix I.

Hadamard transform. As illustrated in Figure 5, Adamas without the Hadamard transform (Adamas w/o Hadamard) directly bucketizes queries and keys before distance-based candidate estimation, achieving near-zero scores across multiple datasets under small token budgets. Its performance improves only gradually with larger budgets and remains below the baseline even at a 4096-token budget. This behavior indicates that the model fails to generate accurate answers without the Hadamard transform, underscoring its critical role in Adamas: Hadamard transform suppresses extreme outliers, and thereby mitigates the information loss introduced by bucketization.

Bucketization. To investigate how bucketization granularity affects accuracy, we compare three variants: 1-bit (Adamas-1bit), 2-bit (Adamas), and 3-bit (Adamas-3bit). As shown in Figure 5, Adamas and Adamas-3bit achieve nearly identical performance, with Adamas-3bit showing only a slight advantage. Adamas-1bit performs comparably to Adamas when the token budget exceeds 256, but its accuracy drops significantly under smaller budgets, suggesting that retaining more bits helps preserve information. However, the marginal improvement of Adamas-3bit over Adamas highlights diminishing returns when increasing bit width. Since higher bit widths also incur greater storage costs, we adopt 2-bit bucketization as a favorable accuracy–storage trade-off.

Distance metrics. We further examine the sensitivity of Adamas to the choice of distance metric by replacing the default ℓ_1 estimator with ℓ_2 distance while keeping all other components unchanged. As shown in Figure 5, the two variants achieve comparable accuracy across datasets and token budgets, with neither metric consistently dominating the other. This result suggests that both metrics provide sufficiently informative signals for candidate selection, and that Adamas is not particularly sensitive to the specific distance function used. We therefore use ℓ_1 throughout the main experiments because it admits a simple implementation over the packed 2-bit representations.

5 Related Work

A large amount of work has explored sparsity in attention mechanisms to improve computational efficiency, with a primary focus on how to effectively select critical KV pairs. Existing methods can be broadly categorized into three groups:

Static sparse methods. These methods adopt predefined sparsity patterns, such as sliding windows or attention sinks. For example, infLLM (Xiao et al., 2024) relies on the sliding window pattern, while StreamingLLM (Xiao et al., 2023) and DUOAttention (Xiao et al., 2025) combine sliding window with attention sink to preserve accuracy. Moreover, DUOAttention requires an extra training phase to determine streaming/retrieval heads, which introduces substantial training overhead. MoA (Fu et al., 2025) further refines this idea by profiling each attention head across layers to adjust the window span and then incorporating attention sinks to build the sparse mask. Although these static designs yield high throughput, their fixed structures risk discarding tokens that are critical to a given query (Tang et al., 2024), often resulting in accuracy degradation under constrained token budgets, particularly on retrieval-intensive tasks.

Dynamic KV selection methods. To better preserve accuracy, these approaches select KV pairs adaptively based on the input query. Quest (Tang et al., 2024) estimates page importance using an upper-bound approximation of attention weights and selects the top-k pages for sparse attention, though its page-level granularity may miss critical tokens. MInference (Jiang et al., 2024) defines three dynamic patterns and performs online estimation per attention head to select the proper pattern, while FlexPrefill (Lai et al., 2025) adaptively determines sparse patterns and selection ratios per head to accelerate long-sequence prefilling. Both methods, however, primarily target the prefilling stage rather than decoding and are therefore complementary to Adamas. We further validate this complementarity by combining Adamas with MInference in Appendix H. SeerAttention (Gao et al., 2024) introduces a learnable gating mechanism that activates important blocks within the attention map, but at the cost of training additional parameters, which complicates deployment by preventing the model from being used out-of-the-box at inference.

Training-based sparse methods. Beyond static and dynamic masking, an alternative approach integrates sparsity into the attention mechanism through specialized training procedures. Reformer (Kitaev et al., 2020) replaces standard dot-product attention with locality-sensitive hashing (LSH), reducing complexity from $O(L^2)$ to $O(L \log L)$. NSA (Yuan et al., 2025) adopts a dynamic hierarchical sparse strategy that combines coarse-grained token compression, fine-grained token selection, and sliding window via gating to balance global context and local precision. MoBA (Lu et al., 2025) incorporates a Mixture-of-Experts (MoE) design (Shazeer et al., 2017) into sparse attention, using a gating mechanism to select historical blocks. While these methods often achieve better accuracy and efficiency than standard full attention, they require pretraining the model, which is computationally costly and limits their applicability. In contrast, our method is training-free and can be directly applied to pretrained LLMs, offering broad compatibility without retraining overhead.

6 Conclusion

In this paper, we present Adamas, an accurate and efficient token-level sparse attention mechanism for long-context inference. Adamas combines the Hadamard transform, bucketization, 2-bit compression, and lightweight distance-based candidate selection to construct compact KV subsets for each decoding query. The compressed representations are used for efficient candidate estimation, while the corresponding original KV states are gathered for sparse attention. Comprehensive evaluations show that Adamas maintains near-lossless performance within the effective context length of Llama-3.1-8B-Instruct. In efficiency evaluations, Adamas achieves up to $6.5\times$ self-attention speedup and $1.7\times$ end-to-end decoding speedup at a 32K context length. Beyond standard long-context benchmarks, Adamas also remains effective on long-form reasoning tasks. With Qwen3-8B in reasoning mode, it achieves performance comparable to full attention on both AIME 2024 and the more recent AIME 2025 benchmark across practical token budgets. These results demonstrate that Adamas provides a practical approach to accelerating long-context decoding while preserving strong task performance across diverse workloads.

AI USE STATEMENT

The core idea, methodology, and algorithmic design of Adamas were developed by the authors. We used generative AI tools to assist with selected implementation and evaluation tasks, including code modification and optimization, experiment monitoring and log inspection, and manuscript editing for clarity and presentation. The tools were also used to help review the presentation of mathematical arguments and summarize or interpret experimental results. All AI-assisted code, analyses, and text were reviewed and verified by the authors. The authors take full responsibility for the final implementation, experiments, results, and claims presented in this paper.

REPRODUCIBILITY STATEMENT

We release the anonymized source code and evaluation scripts for Adamas at the repository linked in Section 4. The release includes the implementation of the Adamas inference pipeline, benchmark evaluation scripts, and efficiency-measurement utilities used in our experiments. Section 3 describes the algorithm and implementation design, while Section 4 specifies the evaluated models, benchmarks, baselines, token budgets, and experimental settings. The appendix provides additional details on bucketization calibration and sensitivity, candidate-selection quality, retrieval and perplexity evaluations, computational and memory-access analysis, detailed LongBench and latency results, ablation studies, and stress tests. Together, the released code and documented experimental settings are intended to support reproduction of the reported accuracy and efficiency results.

REFERENCES

Anthropic. Claude sonnet 4 now supports 1m tokens of context, 2025. https://www.anthropic.com/news/1m-context, Accessed: 2025-09-04.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

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

Yapei Chang, Kyle Lo, Tanya Goyal, and Mohit Iyyer. Booookscore: A systematic exploration of book-length summarization in the era of LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=7Ttk3RzDeu.

Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world models with large language models guided by monte carlo tree search. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=9SpWvX9ykp.

Tri Dao. Fast hadamard transform in cuda, with a pytorch interface, 2024. https://github.com/Dao-AILab/fast-hadamard-transform, Accessed: 2025-09-23.

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

Nelson Elhage, Robert Lasenby, and Christopher Olah. Privileged bases in the transformer residual stream. *Transformer Circuits Thread*, pp. 24, 2023.

Tianyu Fu, Haofeng Huang, Xuefei Ning, Genghan Zhang, Boju Chen, Tianqi Wu, Hongyi Wang, Zixiao Huang, Shiyao Li, Shengen Yan, et al. Mixture of attention spans: Optimizing llm inference efficiency with heterogeneous sliding-window lengths. In *Second Conference on Language Modeling*, 2025.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, et al. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

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

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL https://aclanthology.org/2021.naacl-main.112/.

IBM and Meta. Hadacore: Tensor core accelerated hadamard transform kernel, 2024. https://pytorch.org/blog/hadacore/, Accessed: 2025-09-23.

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

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

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OfjIlbelrT.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph E. Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. How long can open-source llms truly promise on context length?, June 2023. URL https://lmsys.org/blog/2023-06-29-longchat.

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

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

Meta. meta-llama/llama-3.1-8b-instruct, 2024. URL https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct. Accessed: 2026.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=iaYcJKpY2B_.

OpenAI. Introducing gpt-5 for developers, 2025. https://openai.com/index/introducing-gpt-5-for-developers/, Accessed: 2025-09-04.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: query-aware sparsity for efficient long-context llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Minzheng Wang, Longze Chen, Fu Cheng, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, Yunshui Li, Min Yang, Fei Huang, and Yongbin Li. Leave no document behind: Benchmarking long-context LLMs with extended multi-doc QA. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5627–5646, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.322. URL https://aclanthology.org/2024.emnlp-main.322/.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. MMQA: Evaluating LLMs with multi-table multi-hop complex questions. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=GGlpykXDCa.

Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. Collabllm: From passive responders to active collaborators. In *International Conference on Machine Learning (ICML)*, 2025b.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Advances in Neural Information Processing Systems*, 37:119638–119661, 2024.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, junxian guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=cFu7ze7xUm.

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

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

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1126. URL https://aclanthology.org/2025.acl-long.1126/.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

A Bucketization Calibration and Sensitivity

In this section, we analyze the calibration and sensitivity of the bucketization thresholds. We first conduct a lightweight offline calibration using a 10% subsample from each of four representative LongBench datasets, including HotpotQA, TriviaQA, QMSum, and NarrativeQA. Figure 6 shows the empirical distributions of the Hadamard-transformed queries and keys. Both are approximately zero-centered, with most values concentrated around the origin.

Rather than tuning the bucket boundaries independently, we use the empirical standard deviation as a simple scale parameter and define the three thresholds as $\{-\sigma_X, 0, \sigma_X\}$ for the Hadamard vector H_X. The calibration yields $\sigma_Q = 1.35$ and $\sigma_K = 2.26$. Once calibrated, these thresholds are fixed during inference and are not adjusted on a per-example or per-query basis.

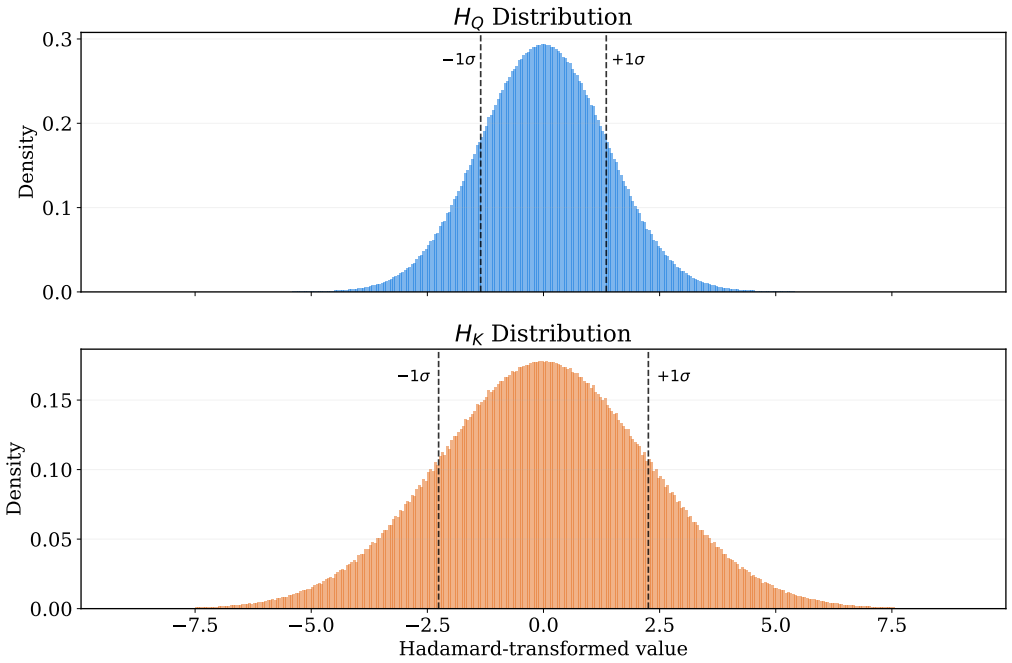


Figure 6: Empirical distributions of the Hadamard-transformed queries H_Q and keys H_K on the calibration subset. Both distributions are approximately zero-centered, with most values concentrated around the origin. The dashed lines indicate $\pm\sigma_X$, which we use as a simple scale-normalized reference for defining the bucketization thresholds.

To evaluate the sensitivity of the bucketization threshold, we vary the threshold multiplier $\alpha \in \{0.5, 1, 1.5, 2\}$, while keeping all other settings fixed. Table 3 reports the results on six LongBench tasks with two model architectures. Performance varies only modestly across this range, and the task-wise optimum does not consistently occur at the same scale. Importantly, $\alpha = 1$ achieves the highest macro-average score on both models. These results suggest that Adamas does not rely on a precisely tuned threshold and that $\alpha = 1$ provides a simple and robust default.

We additionally repeat the calibration using 20% and 50% subsamples from the same datasets and observe nearly unchanged standard deviations, indicating that the calibration statistics can be reliably estimated from a small subset.

B Empirical Analysis of Candidate Selection

We further examine the quality of the candidate sets produced by the distance-based estimator. Since sparse attention ultimately operates on a selected subset of KV tokens, we measure how much of the attention information under full attention is retained by the selected candidates.

Table 3: **Sensitivity analysis of σ-based bucketization thresholds on LongBench.** We vary the threshold multiplier $\alpha \in \{0.5, 1, 1.5, 2\}$ and report results on six LongBench tasks. Best scores for each dataset are in **bold**. The macro-average results indicate that performance is robust to the threshold scale, with 1σ providing a strong default across both models.

(a) Llama-3.1-8B-Instruct

Dataset	0.5σ	1σ	1.5σ	2σ
NarrativeQA	27.87	27.91	28.12	**28.31**
Qasper	**42.88**	42.17	41.12	42.77
MultiFieldQA	53.29	53.76	53.54	**54.15**
HotpotQA	53.22	**54.97**	53.15	53.54
GovReport	**33.19**	33.15	32.55	32.69
TriviaQA	90.91	**91.37**	91.33	90.61
Macro Avg.	50.23	**50.56**	49.97	50.34

(b) LongChat-7b-v1.5-32k

Dataset	0.5σ	1σ	1.5σ	2σ
NarrativeQA	19.13	**19.91**	18.82	18.65
Qasper	**30.40**	30.05	28.83	29.67
MultiFieldQA	41.95	**42.45**	41.13	40.32
HotpotQA	32.24	**32.40**	31.72	32.35
GovReport	**30.91**	30.71	30.51	30.76
TriviaQA	83.87	**83.88**	83.06	82.80
Macro Avg.	39.75	**39.90**	39.01	39.09

Evaluation protocol. We conduct the analysis on LongChat-7b-v1.5-32k using six LongBench datasets: Qasper, NarrativeQA, HotpotQA, MultiFieldQA-en, GovReport, and TriviaQA. We randomly sample 10% of each dataset, resulting in 115 examples in total. To isolate the quality of the token selector from differences in generated sequences, all examples are decoded with full attention. Adamas and Quest (Tang et al., 2024) are evaluated off-path on the same queries, keys, and KV cache states produced by this shared decoding trajectory.

We evaluate token budgets of $\{256, 512, 1024, 2048, 4096\}$. For Quest, we use a page size of 16 tokens. Following the Quest configuration, we exclude the first two layers from the selector analysis for both methods. For Adamas, we use the same calibrated bucketization thresholds as in the main experiments, with $\sigma_Q = 1.35$ and $\sigma_K = 2.26$. Statistics are collected over all remaining layers, attention heads, and decoding steps.

For a query, let

$$P_{\text{full}} = \text{softmax}\left(\frac{\mathbf{q}\mathbf{K}^\top}{\sqrt{d}}\right) \tag{12}$$

denote the attention distribution under full attention, and let $\mathcal{S}$ denote the set of tokens selected by a sparse-attention method. We define the *attention mass recall* as

$$R_{\text{mass}}(\mathcal{S}) = \sum_{i \in \mathcal{S}} P_{\text{full},i}. \tag{13}$$

This metric directly measures the fraction of full-attention probability mass retained by the selected candidate set.

We additionally report the recall of high-attention tokens using a top-p reference set. Specifically, we rank tokens by P_{full} and construct the smallest set $\mathcal{T}_p$ whose cumulative attention mass reaches p. The corresponding set recall is

$$R_{\text{set}} = \frac{|\mathcal{S} \cap \mathcal{T}_p|}{|\mathcal{T}_p|}. \tag{14}$$

We report $p = 0.9$ below.

Table 4 shows that Adamas consistently retains more full-attention mass than Quest across all evaluated token budgets. The difference is most pronounced under high sparsity: with a budget of 256 tokens, Adamas retains 88.97% of the attention mass, compared with 83.88% for Quest. As the budget increases, both methods approach full-attention coverage, while Adamas maintains higher recall throughout.

A similar trend is observed for the high-attention token set. At $p = 0.9$, Adamas consistently recalls a larger fraction of the reference tokens across all budgets. In particular, Adamas exceeds 90% mass recall with a 512-token budget, whereas Quest first exceeds this level at 1024 tokens. For top-p set recall, Adamas exceeds 90% at a budget of 1024, while Quest requires 2048 tokens. At these recall levels, Adamas reaches comparable candidate coverage with roughly half the token

Table 4: **Candidate-selection quality on LongBench.** Attention mass recall and top-p set recall ($p = 0.9$) are averaged equally across six datasets. Both Adamas and Quest are evaluated on the same full-attention decoding trajectories. Higher is better.

	Mass Recall (%)		Top-p Set Recall (%)	
Budget	Adamas	Quest	Adamas	Quest
256	**88.97**	83.88	**76.49**	69.68
512	**92.03**	88.54	**83.80**	78.59
1024	**94.70**	92.27	**90.18**	86.22
2048	**96.93**	95.35	**95.11**	92.45
4096	**98.58**	97.77	**98.18**	96.84

Table 5: Passkey retrieval accuracy (%) of Adamas, Quest, SnapKV and StreamingLLM under different token budgets. The best results are in **bold**. The results highlight the limitations of StreamingLLM's sliding window design, the budget sensitivity of Quest and SnapKV, and the robustness of Adamas across most budgets. Due to incompatibility between SnapKV and Yarn-Llama-2-7b-128k, we omit the corresponding results.

Model	Methods	64	128	256	512	1024	2048	4096
Llama-3.1-8B-Instruct (GQA)	StreamingLLM	1%	1%	1%	1%	1%	2%	4%
	Quest	**48%**	68%	91%	**100%**	**100%**	**100%**	**100%**
	SnapKV	21%	34%	49%	53%	72%	86%	95%
	Adamas (Ours)	45%	**91%**	**100%**	**100%**	**100%**	**100%**	**100%**
Qwen3-8B (GQA)	StreamingLLM	1%	1%	1%	1%	1%	2%	4%
	Quest	9%	19%	76%	76%	76%	86%	85%
	SnapKV	1%	29%	69%	92%	97%	**100%**	**100%**
	Adamas (Ours)	**47%**	**91%**	**98%**	**100%**	**100%**	**100%**	**100%**
Yarn-Llama-2-7b-128k (MHA)	StreamingLLM	1%	1%	1%	1%	1%	2%	4%
	Quest	25%	58%	84%	**95%**	**99%**	99%	99%
	SnapKV	—	—	—	—	—	—	—
	Adamas (Ours)	**54%**	**71%**	**87%**	**95%**	98%	**100%**	**100%**

budget required by Quest, highlighting the benefit of token-level selection under high sparsity. These results indicate that token-wise distance-based selection can construct high-quality candidate sets under substantially constrained token budgets.

C Passkey Retrieval Evaluations

Passkey results. We evaluate passkey retrieval on Llama-3.1-8B-Instruct, Qwen3-8B and Yarn-Llama-2-7b-128k with 100k-token inputs. As shown in Table 5, Adamas maintains consistently high accuracy across token budgets and generally outperforms the baselines under constrained budgets.

StreamingLLM performs poorly across all budgets due to its sliding window design, which discards keys outside the retained span. Quest achieves strong accuracy with large budgets but degrades sharply under small budgets ($\leq$ 128 for 100k), as its page-wise partitioning struggles to recall scattered keys. SnapKV also underperforms under limited budgets, indicating instability in highly budget-constrained retrieval.

Overall, these results demonstrate that Adamas can robustly retrieve relevant KV tokens even under severely limited token budgets.

D Perplexity Evaluations

In this section, we further provide results of perplexity evaluation on Llama-3.1-8B-Instruct (GQA), Qwen3-8B (GQA) and LongChat-7b-v1.5-32k (MHA).

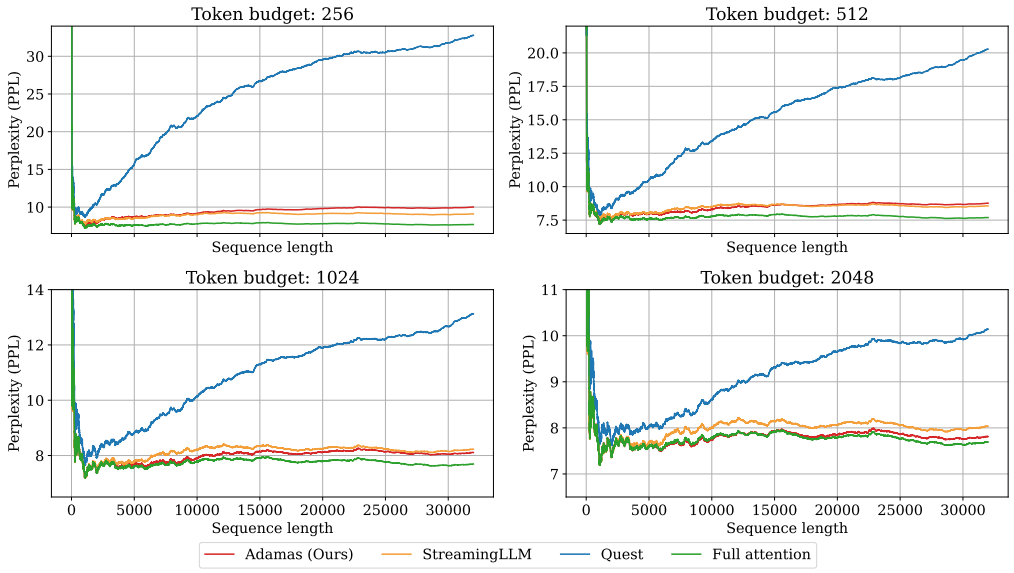


Figure 7: Llama-3.1-8B-Instruct (GQA): perplexity results of StreamingLLM, Quest, Adamas (Ours) and full attention evaluated on PG19 under varying token budgets. Adamas consistently matches full attention and even shows lower perplexity at larger token budget.

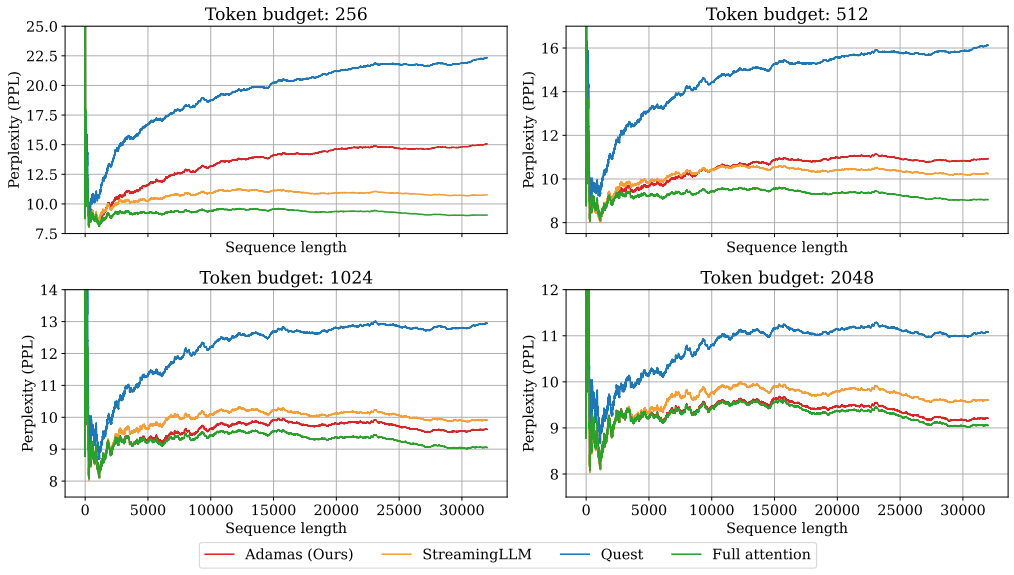


Figure 8: Qwen3-8B (GQA): perplexity results of StreamingLLM, Quest, Adamas (Ours) and full attention evaluated on PG19 under varying token budgets.

As shown in Figure 7, Figure 8 and Figure 9, Adamas closely tracks full attention across all settings, demonstrating its accuracy and robustness. In comparison, Quest yields substantially higher perplexity, as its page-wise strategy introduces many irrelevant KV pairs and can obscure truly relevant ones when the token budget is limited. StreamingLLM maintains stable but consistently higher perplexity due to its eviction strategy, which permanently discards past information. Since SnapKV requires prefilling a long-context input to activate and is therefore incompatible with perplexity evaluation, we omit its result.

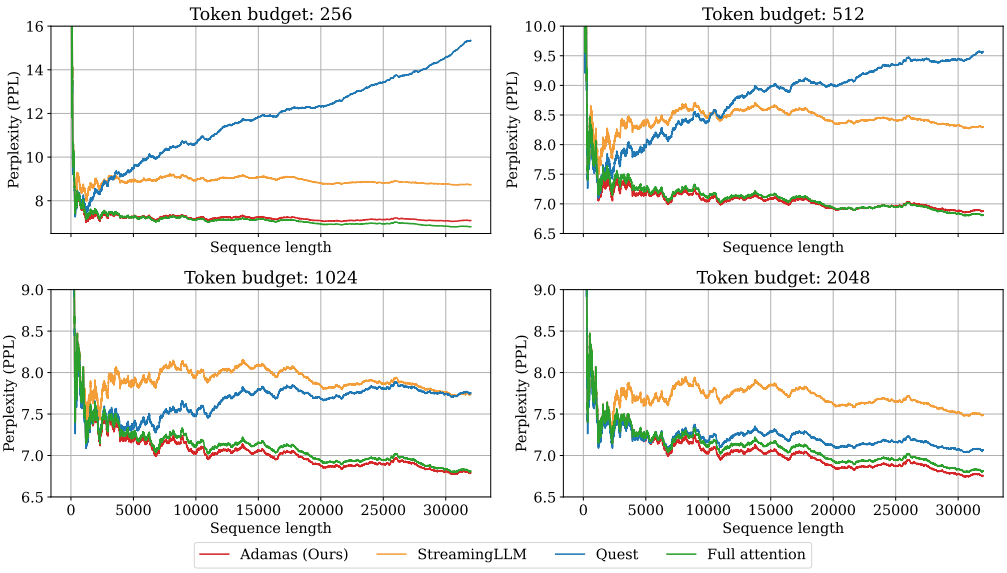


Figure 9: LongChat-7b-v1.5-32k (MHA): perplexity results of StreamingLLM, Quest, Adamas (Ours) and full attention evaluated on PG19 under varying token budgets.

E Computational Workloads and Memory Access Analysis

E.1 Mathematical notations

Table 6: Mathematical notations.

Symbol	Description.
b	Batch size
s	Sequence length
h	Hidden dimension
h_{kv}	Hidden dimension of Key and Value
d	Head dimension
n	Number of buckets
P	Page size
k	Number of selected tokens

E.2 Computational workload

Detailed breakdown of computational workload is shown in Table 7. In summary:

- Computational workload of full attention: $4(h + h_{kv} + s)bh$ FLOPs.
- Computational workload of Quest: $[4h + 4h_{kv} + (\frac{4}{P} + \frac{4log_2(\frac{k}{P})}{Pd})s + 4k] \cdot bh + 2Ph$ FLOPs.
- Computational workload of Adamas: $[4h + 4h_{kv} + 2n + 2 + (\frac{4log_2(k)}{d} + 3)s + 4k] \cdot bh$ FLOPs.

E.3 Memory access

Detailed breakdown of memory access is shown in Table 8. In summary:

- Memory access of full attention:
  - Read access: $3bh + 2bsh_{kv} + 2h^2 + 2hh_{kv} + h + h_{kv}$
  - Write access: $3bh + 2bh_{kv}$

Table 7: Computational workload

	Component	FLOPs
Full attention	QKV projection	$2bh(h+2h_{kv})$
	$A=\mathrm{softmax}(QK^\top)$	$2bsh$
	AV	$2bsh$
	Output projection	$2bh^2$
Quest	QKV projection	$2bh(h+2h_{kv})$
	Reduce keys	$2Ph$
	QK element-wise product	$\frac{2}{P}\cdot bsh$
	Per channel max	$\frac{1}{P}\cdot bsh$
	Page sum	$\frac{1}{P}\cdot bsh$
	Top-k	$\frac{4log_2(\frac{k}{P})}{Pd}\cdot bsh$
	$A=\mathrm{softmax}(QK^\top)$	$2bkh$
	AV	$2bkh$
	Output projection	$2bh^2$
Adamas	QKV projection	$2bh(h+2h_{kv})$
	Hadamard transform	$2bh$
	Bucketization	$2n\cdot bh$
	ℓ_1 distance estimation	$3bsh$
	Top-k	$\frac{4log_2(k)}{d}\cdot bsh$
	$A=\mathrm{softmax}(QK^\top)$	$2bkh$
	AV	$2bkh$
	Output projection	$2bh^2$

Table 8: Memory access

	Component	Read access	Write access
Full attention	QKV projection	$bh+h(h+2h_{kv})+(h+2h_{kv})$	$b(h+2h_{kv})$
	$A=\mathrm{softmax}(QK^\top)V$	$bh+2bsh_{kv}$	bh
	Output projection	$bh+h^2$	bh
Quest	QKV projection	$bh+h(h+2h_{kv})+(h+2h_{kv})$	$b(h+2h_{kv})$
	Reduce keys	$3bh$	$2Ph$
	Criticality estimation	$\frac{2bsh}{P}+bh$	$\frac{bsh}{Pd}$
	Top-k	$\frac{bsh}{Pd}$	$\frac{bkh}{Pd}$
	$A=\mathrm{softmax}(QK^\top)V$	$bh+2bkh_{kv}$	bh
	Output projection	$bh+h^2$	bh
Adamas	QKV projection	$bh+h(h+2h_{kv})+(h+2h_{kv})$	$b(h+2h_{kv})$
	Hadamard transform	$2bh+2hd^2$	$2bh$
	Bucketization	$2bh$	$\frac{bh}{4}$
	ℓ_1 distance estimation	$\frac{bh+bsh}{8}$	$\frac{bsh}{d}$
	Top-k	$\frac{bsh}{d}$	$\frac{bkh}{d}$
	$A=\mathrm{softmax}(QK^\top)V$	$bh+2bkh_{kv}$	bh
	Output projection	$bh+h^2$	$bh+h^2$

- Memory access of Quest:
  - Read access: $7bh+\frac{2d+1}{Pd}\cdot bsh+2bkh_{kv}+2h^2+2hh_{kv}+h+2h_{kv}$
  - Write access: $(5+\frac{s+k}{Pd})bh+2bh_{kv}$

- Memory access of Adamas:
  - Read access: $\frac{57}{8}bh + \frac{bsh}{d} + 2bkh_{kv} + 2d^2 + 2h^2 + 2hh_{kv} + h + 2h_{kv}$
  - Write access: $(\frac{21}{4} + \frac{s+k}{d})bh + 2bh_{kv}$

F LONGBENCH EVALUATIONS

We present detailed data of Table 1 in Table 9, Table 10 and Table 11 for reference. Across the three models, Adamas generally achieves the strongest performance among sparse methods, particularly under limited token budgets.

Table 9: Llama-3.1-8B-Instruct: detailed LongBench evaluation results of StreamingLLM, SnapKV, Quest and Adamas (Ours). The best results are in **bold**, and the second-best results are underlined.

Datasets	Methods / Budget	64	128	256	512	1024	2048	4096
GovReport (34.51)	StreamingLLM	17.20	18.65	20.32	23.26	25.20	28.25	30.86
	SnapKV	16.79	19.54	21.93	24.31	26.51	29.34	31.93
	Quest	13.73	20.67	28.01	32.38	**33.84**	34.23	**34.90**
	Adamas (Ours)	**25.01**	**28.47**	**30.97**	**32.84**	33.29	**34.24**	34.42
HotpotQA (55.65)	StreamingLLM	**45.78**	46.84	45.79	45.66	47.77	48.36	51.28
	SnapKV	31.77	38.34	43.72	46.04	50.77	**55.12**	55.10
	Quest	13.33	26.29	33.23	41.93	51.53	51.70	53.32
	Adamas (Ours)	43.73	**48.84**	**54.78**	**54.66**	**54.66**	54.81	**55.93**
MultifieldQA (55.22)	StreamingLLM	30.58	32.27	32.82	33.78	36.47	38.93	45.45
	SnapKV	24.85	33.20	35.61	39.03	45.42	49.95	53.86
	Quest	15.88	31.00	43.19	51.32	**54.51**	53.69	54.10
	Adamas (Ours)	**33.74**	**46.79**	**50.17**	**53.94**	54.36	**55.41**	**54.94**
NarrativeQA (29.96)	StreamingLLM	**23.16**	21.68	22.67	24.48	24.40	25.30	27.09
	SnapKV	11.18	13.15	14.34	20.07	20.10	24.76	26.81
	Quest	4.51	9.52	14.78	19.89	23.43	26.28	26.52
	Adamas (Ours)	13.20	**23.77**	**24.02**	**28.20**	**28.81**	**28.57**	**29.93**
Qasper (43.18)	StreamingLLM	20.83	20.15	22.41	23.94	28.42	34.56	41.64
	SnapKV	11.39	16.27	19.66	26.04	34.28	40.57	**42.98**
	Quest	10.12	17.15	31.26	35.67	39.08	42.42	41.84
	Adamas (Ours)	**21.16**	**32.16**	**40.26**	**42.77**	**42.55**	**42.69**	42.59
TriviaQA (92.12)	StreamingLLM	83.17	84.95	88.00	88.32	89.58	91.81	91.78
	SnapKV	66.86	71.23	80.04	86.67	90.09	91.19	91.88
	Quest	21.47	46.68	63.42	74.64	80.75	87.47	91.26
	Adamas (Ours)	**83.40**	**89.28**	**88.44**	**91.86**	**92.34**	**92.27**	**91.95**

Table 10: Qwen3-8B: detailed LongBench evaluation results of StreamingLLM, SnapKV, Quest and Adamas (Ours). The best results are in **bold**, and the second-best results are underlined.

Datasets	Methods / Budget	64	128	256	512	1024	2048	4096
GovReport (33.38)	StreamingLLM	14.27	15.62	18.86	22.39	25.81	28.22	31.37
	SnapKV	13.83	16.79	20.42	23.96	26.65	29.76	31.75
	Quest	8.47	13.69	21.79	28.60	32.24	32.75	32.81
	Adamas (Ours)	**18.88**	**29.38**	**35.07**	**34.43**	**33.98**	**33.57**	**33.69**
HotpotQA (53.65)	StreamingLLM	**36.72**	37.77	38.00	38.97	40.60	44.17	49.09
	SnapKV	27.37	**37.90**	36.91	41.84	47.79	50.28	**54.26**
	Quest	4.69	8.09	23.35	32.77	44.52	48.39	50.91
	Adamas (Ours)	14.90	**37.90**	**46.12**	**56.42**	**55.22**	**56.45**	53.10
MultifieldQA (53.09)	StreamingLLM	24.45	29.20	29.23	30.57	33.87	37.55	43.05
	SnapKV	21.22	27.96	33.96	38.29	43.85	46.49	49.99
	Quest	11.85	22.24	33.86	41.18	45.98	48.48	51.25
	Adamas (Ours)	**39.22**	**46.30**	**50.49**	**50.18**	**51.86**	**52.02**	**52.55**
NarrativeQA (24.83)	StreamingLLM	**16.78**	16.71	17.42	18.01	19.95	20.92	22.18
	SnapKV	8.50	9.30	12.64	14.86	16.96	19.78	22.68
	Quest	1.75	4.03	8.07	13.48	18.16	19.59	20.49
	Adamas (Ours)	5.67	**17.15**	**20.49**	**22.12**	**24.22**	**23.14**	**24.88**
Qasper (44.72)	StreamingLLM	**23.63**	24.84	27.19	29.22	31.48	36.99	41.65
	SnapKV	10.86	16.39	18.31	24.06	31.61	38.90	44.00
	Quest	6.62	15.28	24.43	30.07	36.36	39.91	42.21
	Adamas (Ours)	21.42	**32.20**	**38.49**	**42.32**	**44.16**	**45.13**	**44.87**
TriviaQA (88.59)	StreamingLLM	**75.22**	**79.77**	83.60	86.37	87.71	88.12	**88.71**
	SnapKV	29.52	63.39	78.99	84.30	86.89	**88.54**	88.54
	Quest	8.95	18.57	36.50	54.53	68.55	79.24	86.57
	Adamas (Ours)	16.9	59.55	**83.80**	**87.34**	**88.95**	88.46	88.54

Table 11: LongChat-7b-v1.5-32k: detailed LongBench evaluation results of StreamingLLM, SnapKV, Quest and Adamas (Ours). The best results are in **bold**, and the second-best results are underlined.

Datasets	Methods / Budget	64	128	256	512	1024	2048	4096
GovReport (31.08)	StreamingLLM	1.24	1.28	1.18	1.21	6.18	24.80	27.77
	SnapKV	1.55	1.21	3.66	21.11	23.62	26.31	28.47
	Quest	3.30	11.93	22.84	27.27	29.89	**31.19**	**31.23**
	Adamas (Ours)	**30.41**	**30.44**	**30.32**	**30.25**	**31.03**	31.10	30.86
HotpotQA (31.01)	StreamingLLM	10.72	12.02	13.90	10.20	14.56	24.80	27.04
	SnapKV	10.55	12.16	20.28	**35.65**	**33.90**	**35.23**	**33.94**
	Quest	5.31	15.49	21.54	26.64	30.29	32.21	32.93
	Adamas (Ours)	**31.18**	**31.89**	**31.39**	33.76	32.45	32.82	31.58
MultifieldQA (41.96)	StreamingLLM	4.14	5.88	4.81	3.95	11.30	28.78	33.63
	SnapKV	7.53	6.24	16.74	40.98	37.76	40.03	42.82
	Quest	6.99	20.87	31.12	39.80	42.03	**44.09**	**43.25**
	Adamas (Ours)	**36.60**	**40.67**	**41.05**	**44.10**	**42.76**	41.98	42.38
NarrativeQA (21.53)	StreamingLLM	3.67	3.47	4.91	2.69	5.45	16.48	19.09
	SnapKV	4.62	2.45	9.50	17.97	19.47	19.31	**20.32**
	Quest	1.86	5.98	14.2	16.28	17.91	19.66	19.88
	Adamas (Ours)	**17.81**	**18.52**	**18.98**	**19.82**	**19.52**	**20.71**	20.53
Qasper (28.74)	StreamingLLM	7.51	8.03	7.43	6.73	9.56	21.90	28.41
	SnapKV	11.11	8.97	9.02	25.25	26.40	28.92	29.25
	Quest	8.67	18.54	26.72	**30.89**	**31.01**	**31.86**	**29.79**
	Adamas (Ours)	**28.33**	**29.38**	**31.30**	30.49	30.39	29.18	28.89
TriviaQA (83.55)	StreamingLLM	2.65	3.09	3.84	9.86	41.94	82.42	83.28
	SnapKV	0.67	9.19	55.92	**84.01**	80.88	80.12	81.25
	Quest	12.29	44.68	68.92	81.32	**83.95**	**85.84**	**84.94**
	Adamas (Ours)	**78.91**	**82.95**	**83.65**	83.80	83.80	84.06	84.11

G End-to-end Efficiency Evaluation

In this section, we present detailed results of Figure 4.

On A6000, Quest is slightly faster than Adamas on LongChat-32k, while Adamas still achieves approximately $1.5\times$–$1.7\times$ speedup over full attention. On Llama-3.1-32k, SnapKV achieves the lowest latency, whereas Quest is not applicable.

On H800, Adamas exhibits more stable performance across settings. For LongChat-48k and LongChat-56k, Adamas is either the fastest or close to the fastest among the evaluated methods, achieving approximately $1.6\times$–$1.8\times$ speedup over full attention, while SnapKV generally incurs higher latency. For Llama-3.1, Adamas is the only method with complete results at 56k context length, whereas both Quest and SnapKV have missing results. This indicates that Adamas provides better usability and robustness under longer-context settings.

Table 12: Decoding latency (ms) measured on A6000. "–" denotes OOM failures.

Context length	Model	Method	256	512	1024	2048	4096	Full
32k	LongChat-7b-v1.5-32k	Quest	25.95	26.17	26.55	27.48	28.65	47.28
		SnapKV	32.85	33.29	34.25	36.07	39.90	
		Adamas (Ours)	27.91	27.96	28.42	29.24	30.82	
32k	Llama-3.1-8B-Instruct	Quest	–	–	–	–	–	32.35
		SnapKV	19.33	19.44	20.10	21.29	23.57	
		Adamas (Ours)	28.88	28.86	29.06	29.25	29.96	

Table 13: Decoding latency (ms) measured on H800. "–" denotes OOM failures.

Context length	Model	Method	256	512	1024	2048	4096	Full
48k	LongChat-7b-v1.5-32k	Quest	14.67	14.66	14.93	15.21	15.73	24.37
		SnapKV	17.75	18.17	18.95	20.17	22.60	
		Adamas (Ours)	14.23	14.27	14.51	14.95	15.77	
48k	Llama-3.1-8B-Instruct	Quest	16.22	16.13	16.53	16.67	17.45	15.68
		SnapKV	19.06	19.28	19.81	21.25	–	
		Adamas (Ours)	14.70	14.68	14.87	15.06	15.27	
56k	LongChat-7b-v1.5-32k	Quest	15.32	15.28	15.47	15.90	16.44	26.79
		SnapKV	18.05	18.25	18.98	20.03	22.59	
		Adamas (Ours)	14.70	14.91	15.21	15.66	16.33	
56k	Llama-3.1-8B-Instruct	Quest	–	–	–	–	–	16.49
		SnapKV	–	–	–	–	–	
		Adamas (Ours)	15.19	15.27	15.26	15.53	15.98	

H Combining Adamas with MInference

Adamas primarily accelerates the autoregressive decoding stage, while existing sparse-attention methods for prefilling optimize a different phase of inference. To examine whether these optimizations can be combined, we integrate Adamas with MInference (Jiang et al., 2024), using MInference during prefilling and Adamas during decoding. This configuration requires no additional training or architectural modification: MInference is enabled for prompt-side attention computation, and the model switches to Adamas for token-wise KV selection during autoregressive decoding.

We evaluate the combined pipeline at context lengths of 100K and 128K across token budgets from 256 to 4096. We additionally include the full attention setting for reference. Table 14 summarizes the results averaged across token budgets, and Table 15 shows detailed results.

As shown in Table 14, combining MInference with Adamas substantially reduces prefill latency while leaving decoding latency nearly unchanged. Averaged across all evaluated settings, MInfer-

Table 14: **Latency of Adamas combined with MInference.** MInference is applied during prefilling, while Adamas is used during decoding. Results are averaged across all evaluated settings. The relative change is measured against Adamas without MInference.

	Prefill (s)			Decode (s/token)		
Setting	Adamas	+MInference	Change	Adamas	+MInference	Change
100K	16.656	11.895	-28.6%	0.016692	0.016768	$+0.45\%$
128K	25.794	17.269	-33.0%	0.018559	0.018661	$+0.56\%$
Average	21.225	14.582	$\mathbf{-30.8\%}$	0.017625	0.017714	$+0.51\%$

ence reduces prefill latency from 21.225 to 14.582, corresponding to a 30.8% reduction, or approximately a $1.46\times$ speedup. The improvement becomes slightly larger at longer contexts, reaching 33.0% at 128K compared with 28.6% at 100K.

In contrast, the decoding latency changes by only 0.51% on average. This is expected because MInference operates only during the prefill stage, while decoding continues to use the same Adamas candidate-selection and sparse-attention pipeline. These results show that the two methods provide complementary acceleration across different inference stages: MInference reduces prompt-processing cost, whereas Adamas reduces the repeated attention cost during autoregressive decoding.

Table 15: **Detailed latency (s for prfill and s/token for decode) results for combining Adamas with MInference.** MInference accelerates the prefill stage, while the decoding stage continues to use Adamas. Δ reports the relative latency change after enabling MInference.

Token Budget	Context Length	Adamas Prefill	+MInference Prefill	Δ Prefill	Adamas Decode	+MInference Decode	Δ Decode
256	100K	16.544	11.755	-28.9%	0.016001	0.016073	$+0.45\%$
256	128K	25.769	17.316	-32.8%	0.017860	0.017976	$+0.65\%$
512	100K	16.654	11.888	-28.6%	0.016053	0.016153	$+0.62\%$
512	128K	25.783	17.297	-32.9%	0.017784	0.017936	$+0.86\%$
1024	100K	16.687	11.890	-28.7%	0.016207	0.016331	$+0.76\%$
1024	128K	25.771	17.229	-33.1%	0.018091	0.018224	$+0.74\%$
2048	100K	16.669	11.945	-28.3%	0.016609	0.016546	-0.38%
2048	128K	25.777	17.275	-33.0%	0.018322	0.018530	$+1.14\%$
4096	100K	16.670	11.921	-28.5%	0.017038	0.017128	$+0.53\%$
4096	128K	25.760	17.301	-32.8%	0.018947	0.018891	-0.29%
Full	100K	16.710	11.970	-28.4%	0.018244	0.018377	$+0.73\%$
Full	128K	25.903	17.197	-33.6%	0.020350	0.020407	$+0.28\%$

As shown in Table 15, the reduction in prefill latency is consistent across all token budgets, ranging from 28.3%–28.9% at 100K and 32.8%–33.6% at 128K. Meanwhile, the decoding differences remain within approximately $\pm 1\%$ for nearly all settings, confirming that enabling MInference does not materially affect the Adamas decoding pipeline.

Overall, Adamas can be naturally combined with prefill-oriented sparse attention methods such as MInference, extending its decoding-time acceleration to a more complete long-context inference pipeline without additional training.

I ABLATION STUDIES

Below we present detailed ablation results evaluated on LongBench with LongChat-7b-v1.5-32k in Table 16.

Table 16: Ablation results evaluated on LongBench with LongChat-7b-v1.5-32k.

Datasets	Methods / Budget	16	32	64	128	256	512	1024	2048	4096
GovReport (31.12)	Adamas (Ours)	22.08	28.31	**30.41**	30.44	30.37	**31.18**	30.77	**31.00**	31.07
	Adamas w/o Hadamard	0.60	0.59	0.59	0.64	2.06	6.32	14.12	23.53	29.14
	Adamas-1bit	17.17	23.67	27.88	30.09	30.72	31.16	**31.38**	30.84	30.70
	Adamas-3bit	**24.69**	**29.01**	30.05	**30.59**	30.83	31.01	30.95	30.71	**31.33**
	Adamas w/ L2 distance	18.27	27.19	30.09	30.45	**30.86**	30.93	30.96	30.50	30.56
HotpotQA (31.07)	Adamas (Ours)	24.05	**31.65**	32.18	32.89	32.30	31.45	33.06	**32.56**	31.41
	Adamas w/o Hadamard	1.01	1.04	0.68	0.48	0.77	2.07	5.38	13.47	25.13
	Adamas-1bit	19.22	24.70	29.04	31.55	**33.79**	31.63	32.14	32.47	31.58
	Adamas-3bit	**27.72**	31.17	31.29	31.82	32.56	**32.12**	**33.25**	32.21	**31.61**
	Adamas w/ L2 distance	18.73	28.83	**32.58**	**32.91**	31.15	30.70	32.66	31.86	31.10
MultifieldQA (41.64)	Adamas (Ours)	21.53	33.38	36.60	40.67	42.49	**42.93**	42.22	40.77	41.83
	Adamas w/o Hadamard	2.97	2.84	2.51	5.27	8.28	18.26	27.45	39.24	41.29
	Adamas-1bit	23.21	29.51	35.64	38.98	40.53	41.99	42.43	41.62	42.01
	Adamas-3bit	**30.04**	**36.00**	36.28	40.54	41.96	42.73	**43.14**	41.17	41.86
	Adamas w/ L2 distance	23.70	34.11	**39.81**	**41.41**	**43.08**	41.60	41.28	**41.65**	**42.33**
NarrativeQA (21.23)	Adamas (Ours)	11.98	15.13	**17.81**	**18.52**	17.36	18.59	18.74	20.14	20.22
	Adamas w/o Hadamard	1.68	1.00	1.12	0.91	0.38	1.34	3.29	6.00	13.32
	Adamas-1bit	5.38	9.16	14.79	15.18	17.34	**19.39**	**19.94**	20.00	20.01
	Adamas-3bit	**12.61**	**15.48**	17.65	16.73	17.91	19.05	19.03	**20.35**	20.09
	Adamas w/ L2 distance	8.83	14.50	16.06	18.39	**18.37**	19.20	18.54	19.74	**20.33**
Qasper (28.89)	Adamas (Ours)	18.83	24.52	28.33	29.38	**30.68**	30.93	29.94	29.26	28.65
	Adamas w/o Hadamard	1.73	2.40	2.29	4.93	9.24	18.27	26.17	**31.08**	**30.14**
	Adamas-1bit	18.04	20.81	24.16	28.73	29.50	30.04	30.18	30.52	28.73
	Adamas-3bit	21.95	26.07	26.39	30.23	30.52	31.12	29.50	29.51	28.81
	Adamas w/ L2 distance	**22.41**	**27.04**	**28.70**	**31.36**	30.35	**31.23**	**30.80**	29.36	28.86
TriviaQA (84.25)	Adamas (Ours)	56.77	75.13	78.91	82.95	**84.67**	83.99	83.36	83.75	83.95
	Adamas w/o Hadamard	2.63	2.08	1.21	3.52	9.73	20.14	33.15	56.62	79.13
	Adamas-1bit	52.26	70.62	80.08	80.88	82.59	83.90	**84.11**	**84.22**	83.63
	Adamas-3bit	**70.99**	**77.74**	**81.40**	82.95	83.99	83.79	83.46	83.55	83.68
	Adamas w/ L2 distance	37.82	66.80	80.97	**83.35**	84.21	**84.29**	83.60	84.03	**84.25**

J NEEDLE-IN-A-HAYSTACK STRESS TESTS

We conduct the Needle-in-a-Haystack (NIAH) stress tests using Yarn-Llama-2-7b-128k model to evaluate the recall rates of Adamas, Quest, and full attention at a context length of 32k. As shown in Figure 10, Adamas maintains a recall performance that is nearly identical to that of full attention even under high sparsity (with only a 256-token budget), consistently outperforming the baseline. Specifically, both Adamas and full attention achieve scores around 7/10 at a sequence length of 32k, whereas Quest reaches only a little over 2/10. These results further demonstrate the strong retrieval capability of Adamas under high sparsity.



Figure 10: NIAH stress test on Adamas (ours), Quest and full attention, using Yarn-Llama-2-7b-128k. Token budgets range from 256 to 2048. Adamas consistently outperforms Quest in all budgets.