AB-Sparse: SPARSE ATTENTION WITH ADAPTIVE BLOCK SIZE FOR ACCURATE AND EFFICIENT LONG-CONTEXT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Block sparse attention makes long-context decoding affordable by partitioning the KV cache into fixed-size blocks and loading only the blocks a query needs, and it has become the most deployable form of sparse attention because it leaves paged KV cache management untouched. Its weakness is accuracy: at a 4% KV budget, state-of-the-art methods trail full attention by up to 10 points on standard long context benchmarks. We trace a large part of this gap to a single design choice, the use of one block size for every attention head. Measuring attention recall head by head, we find that heads differ widely in their sensitivity to block granularity: some keep near-perfect recall, others lose over 90% of it when blocks grow from 16 to 32 tokens, and no single size serves both. We present *AB-Sparse*, a training-free algorithm–system co-design that gives each head its own block size. A lightweight one-time calibration assigns block sizes that transfer across domains; INT4 per-channel asymmetric quantization of block centroids, which only rank blocks and never enter the attention output, keeps centroid memory below that of the uniform baseline; and custom GPU kernels batch heads with heterogeneous block counts without padding and map variable-size blocks onto standard paged attention. On Llama-3.1-8B, Qwen3-8B, and Qwen3-32B, *AB-Sparse* raises the accuracy of Quest and ArkVale by 3.0–3.7 points on RULER and 1.9–2.6 points on LongBench on average (up to 5.4 points), recovering 27–85% of the gap to full attention, with comparable or higher decoding throughput.

1 INTRODUCTION

Large language models (LLMs) Brown et al. (2020); Liu et al. (2024a); Yang et al. (2025a) are increasingly asked to read entire codebases Luo et al. (2024), document collections Yang et al. (2025b), and their own long chains of reasoning Wei et al. (2022). Every decoding step of such a request must load its entire KV cache, whose size grows linearly with context length: at 128K tokens, the KV cache of a single Llama-3.1-8B Meta (2024) request occupies 16 GB, as much as the model weights themselves. Decoding is therefore bound by the memory traffic of the KV cache rather than by computation.

Attention is fortunately sparse: at each step a small subset of tokens receives most of the attention mass Xiao et al. (2023), so a decoder that loads only those tokens can be much faster; the question is how to find them cheaply. Figure 1 summarizes the three answers proposed so far and the price each pays. **Token-based** methods such as H_2O Zhang et al. (2023) and InfiniGen Lee et al. (2024) score every token at every step, buying accuracy with a selection overhead that itself scales with context length. **Semantic-based** methods such as ClusterKV Liu et al. (2024c) and RetroInfer Chen et al. (2025) cluster keys by similarity and retrieve whole clusters, buying efficiency by restructuring the cache into layouts that standard paged KV cache management Kwon et al. (2023); Zheng et al. (2024); Ye et al. (2025) cannot serve. **Block-based** methods such as Quest Tang et al. (2024) and ArkVale Chen et al. (2024a) split the cache into fixed-size blocks, summarize each block by a centroid, and load only the Top-K blocks; they keep the paged layout untouched and add one small ranking step, which makes them the most deployable of the three. What they give up is accuracy. At a 4% KV budget, Quest and ArkVale trail full attention by 8.0–10.5 and 3.7–8.1 points on RULER across three models (Table 1), and this gap, not throughput, is now the barrier to their wider use.

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.

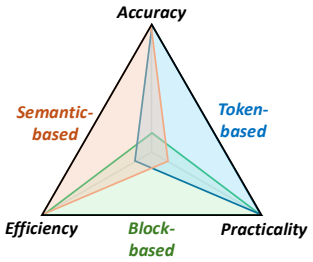


Figure 1: Qualitative comparison of various sparse attention paradigms.

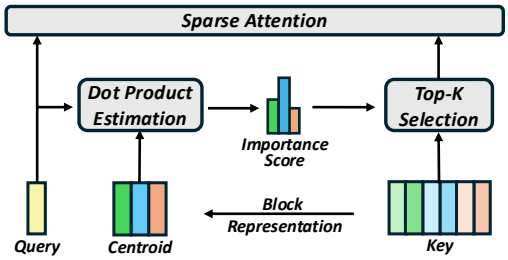


Figure 2: Illustration of block sparse attention workflow.

Where does the gap come from? Block sparse attention selects blocks, so its accuracy is bounded by how well a centroid represents its block, and that depends on the block size. Every existing method, however, fixes one block size for every attention head Tang et al. (2024); Chen et al. (2024a); Xiao et al. (2024a); Lu et al. (2025), even though heads are known to play very different roles Xiao et al. (2024b); Wu et al. (2024). Our measurement study (§2.3) shows that this uniformity is exactly where accuracy is lost: some heads keep near-perfect attention recall at any block size, whereas others lose more than 90% of their recall once blocks grow from 16 to 32 tokens (Figure 3). A single block size therefore serves nobody well. It is too coarse for the sensitive heads, which miss critical tokens, and needlessly fine for the insensitive ones, which pay for centroids they do not need. Giving each head its own block size resolves the tension: in our study, per-head assignment raises the average recall of Llama-3.1-8B and Qwen3-8B from 89.7% and 77.8% to 95% while *increasing* the average block size from 32 to 44.2 and 39.5. The opportunity is thus to gain accuracy for free, provided that adaptive block sizes can be made as cheap to run as uniform ones.

That proviso raises three challenges, each with a concrete cost. Block sizes cannot be tuned at run time, because re-blocking a head means recomputing its centroids from every key in the cache. Finer blocks mean more centroids, whose memory is a $1/B$ fraction of the KV cache and which are read at every decoding step. And heterogeneous block sizes break the two uniformity assumptions on which fast kernels rest, equal centroid counts across heads and a single block-to-page stride, so naive implementations fall back to per-head loops and gather copies that are 3–9$\times$ slower (Figure 13).

We present *AB-Sparse*, a training-free, algorithm–system co-designed framework that turns per-head block sizes into a drop-in replacement for the uniform block size of existing block sparse attention, answering the three challenges with three matching designs. Because a head's sensitivity to block granularity is a property of its learned role rather than of a particular input, *AB-Sparse* determines block sizes *once*, by a lightweight calibration on a small generic corpus whose assignments transfer to unseen domains (§3.2). Because centroids only rank blocks and never enter the attention output, *AB-Sparse* quantizes them to INT4 with a per-channel asymmetric scheme that preserves block ranking and cuts centroid memory by $4\times$, more than paying for the finer blocks (§3.3). Because both uniformity assumptions can be restored at the index level, *AB-Sparse* batches all heads through a prefix-indexed centroid layout without padding and maps variable-size blocks onto fixed-size pages through a stride view, so unmodified paged attention kernels run on top (§3.4).

We evaluate *AB-Sparse* on Llama-3.1-8B, Qwen3-8B, and Qwen3-32B with RULER and LongBench. Applied on top of Quest and ArkVale, it raises the benchmark-average accuracy of both baselines on all three models, by 3.0–3.7 points on RULER and 1.9–2.6 points on LongBench on average (up to 5.4 points), recovering between 27% and 85% of the gap to full attention, while matching or exceeding their decoding throughput. Our contributions are:

- A measurement study showing that attention heads differ widely in their sensitivity to block granularity, and that this sensitivity is a stable, head-specific property that a uniform block size cannot accommodate (§2.3).
- *AB-Sparse*, which makes per-head block sizes practical through calibration-driven allocation, near-lossless INT4 centroid quantization, and GPU kernels that batch heterogeneous heads without padding and reuse standard paged attention (§3).

- An evaluation on three LLMs and two long-context benchmarks showing consistent accuracy gains over uniform-block-size baselines at unchanged KV budget and comparable or higher throughput (§4).

2 Background and Motivation

2.1 Attention and the KV Cache

At decoding step t, each attention head Vaswani et al. (2017); Ainslie et al. (2023) computes scores between the query $\mathbf{q}_t \in \mathbb{R}^{1\times d}$ and the keys $\mathbf{k}_i$ of all preceding tokens, normalizes them with a softmax, and uses the resulting weights $a_{t,i}$ to average the values $\mathbf{v}_i$:

$$z_{t,i} = \frac{\mathbf{q}_t \cdot \mathbf{k}_i^T}{\sqrt{d}}, \quad a_{t,i} = \frac{e^{z_{t,i}}}{\sum_{j=1..t} e^{z_{t,j}}}, \quad \mathbf{o}_t = \sum_{i=1..t} a_{t,i} \cdot \mathbf{v}_i. \tag{1}$$

Keys and values of processed tokens are cached so that decoding costs $O(n)$ per step instead of $O(n^2)$, but every step must then read the whole KV cache, which makes long-context decoding memory-bound. Throughout the paper, h indexes *key–value heads*: under grouped-query attention Ainslie et al. (2023) the query heads that share a KV head share its cache, its block summaries, and, in ***AB-Sparse***, its block size.

2.2 Block Sparse Attention

Only a small subset of tokens dominates the attention output Jiang et al. (2024); Liu et al. (2024b); Chen et al. (2024b), and block sparse attention such as Quest Tang et al. (2024) and ArkVale Chen et al. (2024a) exploits this without touching the paged KV cache layout of serving systems Kwon et al. (2023); Zheng et al. (2024); Ye et al. (2025). As Figure 2 shows, the cache of each head is partitioned into blocks of B tokens, and each block i is summarized by a centroid c_i[*]. At each step the query scores all centroids, $r_{t,i} = \mathbf{q}_t \cdot c_i^\top$, and attention is computed only over the Top-K blocks.

The block size B sets the accuracy–efficiency trade-off of this scheme, and both sides can be quantified. A larger B makes each centroid stand for more tokens, so ranking blocks by centroid becomes a coarser proxy for ranking tokens by attention: in our measurements (§2.3), the recall of the selected tokens drops from 100% at $B = 16$ to below 10% at $B = 32$ for the most sensitive heads. A smaller B multiplies centroids: a block of B tokens holds $2B$ key and value vectors but carries two centroid vectors (e.g., the per-channel minimum and maximum), so centroid memory equals a $1/B$ fraction of the KV cache and must be read at every step, 3.1% of the cache at $B = 32$ and 6.3% at $B = 16$. All existing block sparse methods fix one B for every head Tang et al. (2024); Chen et al. (2024a); Xiao et al. (2024a); Lu et al. (2025) and thereby fix one point on this trade-off for heads that, as we show next, sit at very different points.

2.3 Heads Disagree on the Right Block Size

In some heads the critical tokens cluster in a few contiguous regions, in others they scatter; a centroid represents the former well at any block size and the latter only when blocks are small, so heads should differ in how much recall they lose as B grows. We test this on Llama-3.1-8B Meta (2024) and Qwen3-8B Qwen (2025a) with 32K-token Wikipedia wikipedia (2025) inputs. For every head we vary $B \in \{16, 32, 64\}$ at a fixed token budget of 4096[†] and measure *attention recall*, the fraction of attention mass that falls inside the selected blocks.

Sensitivity to block size is strongly head-specific. Figure 3 plots recall normalized to its value at $B = 16$ for representative heads. Insensitive heads keep normalized recall at 1.0 up to $B = 64$, whereas sensitive heads fall below 0.1 already at $B = 32$; between these extremes lie heads that tolerate 32 but not 64. Heads of all three kinds are common in every layer of both models and interleaved within the same layer.

[*]Centroids can be computed by mean pooling Lu et al. (2025), min–max pooling Tang et al. (2024), or bounding volumes Chen et al. (2024a); the choice is orthogonal to this work.

[†]Top-K is chosen so that the selected blocks cover 4096 tokens; the trend holds for other budgets.

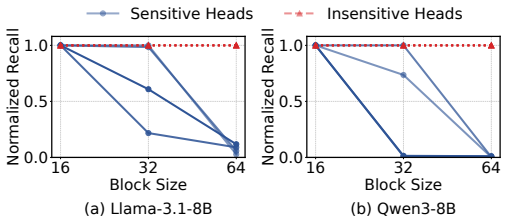


Figure 3: Normalized recall curves across block sizes. Insensitive heads maintain near-perfect normalized recall, while sensitive heads degrade sharply as block size increases.

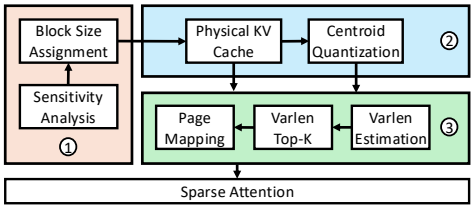


Figure 4: Architecture of *AB-Sparse*, highlighting head-wise block-size assignment, centroid management, and heterogeneous-block attention execution.

A per-head block size dominates any uniform one. With $B = 32$ for all heads, average recall is only 89.7% on Llama-3.1-8B and 77.8% on Qwen3-8B. Assigning each head the largest block size that retains 95% of its peak recall lifts average recall to 95% on both models while the average block size *grows* to 44.2 and 39.5, respectively. Adaptive block sizes thus improve accuracy without decreasing average block size; the rest of this paper makes them practical.

3 *AB-Sparse* DESIGN

3.1 OVERVIEW

Section 2.3 promises a rare trade: serve every head at the block size it needs, and accuracy rises while the centroid count falls. *AB-Sparse* is built to collect on this promise inside an unmodified serving stack, which means per-head block sizes must cost nothing at run time; three costs stand in the way. *Assignment cost*: the block size of a head cannot be changed on the fly, because centroids are computed from the keys and re-blocking a head means re-reading all of its L keys; at 128K tokens this is a pass over the entire KV cache for every candidate size. *Memory cost*: a head at $B = 16$ stores twice the centroids of a head at $B = 32$, and centroids are read at every decoding step, so finer blocks tax both capacity and bandwidth (§2.3). *Execution cost*: with per-head block sizes, head h owns $N_h = \lceil L/B_h \rceil$ centroids and K_h selected blocks, which violates the equal-shape assumption of batched kernels and the single block-to-page stride assumption of paged attention; the naive fallbacks, per-head loops and gather copies, are 3–9$\times$ slower than batched execution (Figure 13).

AB-Sparse removes each cost with one design, summarized in Figure 4. ① It *calibrates once*: because sensitivity to block size is a stable property of a head, a short offline profiling run fixes each head's block size for all future requests (§3.2). ② It *quantizes centroids to INT4*: because centroids only rank blocks, a per-channel asymmetric scheme preserves the ranking while cutting centroid memory and traffic by $4\times$, which more than pays for the finer blocks (§3.3). ③ It *restores uniformity at the index level*: a prefix-indexed centroid layout batches all heads without padding, and a stride view maps blocks of any size onto fixed-size pages, so standard paged attention kernels run unchanged (§3.4).

3.2 CALIBRATION-DRIVEN BLOCK SIZE ALLOCATION

Why one calibration suffices. Heads acquire specialized roles during training, such as local pattern matching or long-range retrieval Xiao et al. (2024b); Wu et al. (2024); Jiang et al. (2024); Liu et al. (2026), and a head's role, not the input, determines how its attention mass is spread over the cache and hence how much recall it loses when blocks grow. Block size sensitivity should therefore be a per-head constant, measurable once on any sufficiently diverse text, and two observations confirm this. First, assignments calibrated on 128 Wikipedia wikipedia (2025) samples transfer to code, QA, and summarization inputs from LongBench Bai et al. (2024): as Figure 5 shows, at the same average block size they recover about half of the recall that uniform $B = 32$ loses relative to $B = 16$ and come within 0.03 recall of an *oracle* calibrated on the target domain itself. Second, the same assignments improve accuracy at every context length from 16K to 96K and on every benchmark average of three models (Table 1), including tasks whose distribution differs sharply from Wikipedia.

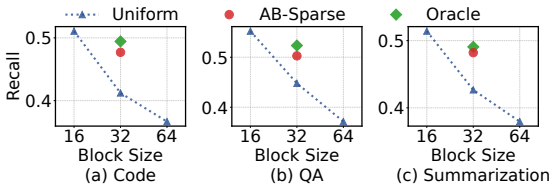


Figure 5: Average recall comparison between adaptive and uniform block size. Despite the adaptive assignments are calibrated solely on Wikipedia wikipedia (2025); they consistently outperform uniform block size across code, QA, and summarization tasks and nearly match an in-domain oracle.

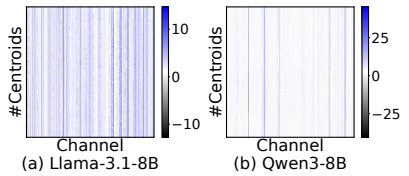


Figure 6: Centroid value distribution of Llama-3.1-8B and Qwen3-8B. The column-wise patterns indicate that centroid values are tightly clustered per channel, supporting the use of per-channel quantization.

Procedure. For each head, calibration measures the attention recall $\mathrm{Recall}(h, B)$ under each candidate block size on the calibration samples and selects the largest block size that retains a fraction τ of the recall achieved at the finest size:

$$B_h^* = \max\{B \mid \mathrm{Recall}(h, B) \geq \tau \cdot \mathrm{Recall}(h, B_{\min})\}, \tag{2}$$

where $B_{\min}$ is the smallest candidate. The threshold τ is the single knob of the method: a higher τ assigns finer blocks to more heads and a lower one tolerates a small recall loss in exchange for fewer centroids. We use $\tau = 0.95$ and candidates $\{16, 32, 64\}$ throughout, which places the arithmetic mean block size close to the uniform baseline's 32. Calibration is a single forward pass over the calibration set and produces one integer per head, so it adds no run-time work.

Resource accounting. Head h owns $\lceil L/B_h \rceil$ centroids, so the centroid memory of a model is a $\frac{1}{H}\sum_h 1/B_h$ fraction of its KV cache, i.e., the inverse of the *harmonic* mean block size, which is smaller than the arithmetic mean whenever block sizes differ. The INT4 quantization of §3.3 makes this irrelevant in practice: even if every head were assigned $B = 16$, INT4 centroids would occupy $6.3\%/4 \approx 1.6\%$ of the KV cache, half of the 3.1% that the uniform $B = 32$ baseline spends on BF16 centroids, so *AB-Sparse* never stores or reads more centroid bytes than the baseline it replaces.

3.3 Near-lossless Centroid Quantization

Centroids are at once the most expendable and the most exposed data in block sparse attention: expendable because they only rank blocks and never enter the attention output, so a value error that leaves the Top-K set unchanged costs nothing, and exposed because every one of them is read at every decoding step, so each of their bytes is paid for at every token. Quantization suits the first property and is demanded by the second; the only question is how far precision can drop before rankings change.

Figure 6 shows that, along each channel of the head dimension, centroid values of different blocks are tightly clustered: the distribution is column-wise, not row-wise. A per-channel scale therefore captures the range of a channel with little error, and, because the clusters are not centered at zero, an asymmetric zero point captures their offset. We compare bit widths and schemes by their effect on the quantity that matters, Top-K page recall (Figure 7). INT2 destroys the ranking, with recall of 0.3–0.7 across the layers of Llama-3.1-8B. INT4 with symmetric per-tensor scaling recovers most of it (about 0.85) but keeps a visible gap. INT4 with asymmetric per-channel scaling holds recall at 0.9–0.95 in every layer, within 0.1 of INT8, whose recall is indistinguishable from BF16. Since INT4 quarters centroid memory where INT8 only halves it, and Figure 12 shows that INT4 leaves end-task accuracy unchanged, *AB-Sparse* adopts INT4 asymmetric per-channel quantization and dequantizes inside the estimation kernel.

3.4 Custom GPU Kernels for Heterogeneous Blocks

Heterogeneous block sizes break two uniformity assumptions of fast attention kernels, equal centroid counts across heads and a single block-to-page stride, and *AB-Sparse* restores both at the index level so that nothing is padded, gathered, or copied. A naive implementation gives up one assumption or

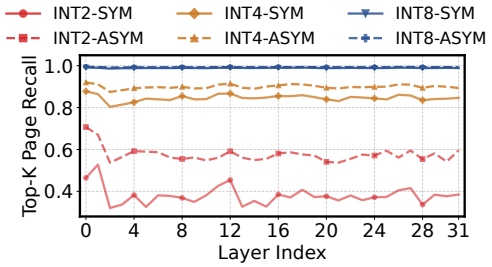


Figure 7: Top-K page recall across layers on Llama-3.1-8B under different quantization bit widths and strategies. INT4 asymmetric per-channel quantization consistently maintains recall above 0.9 across all layers.

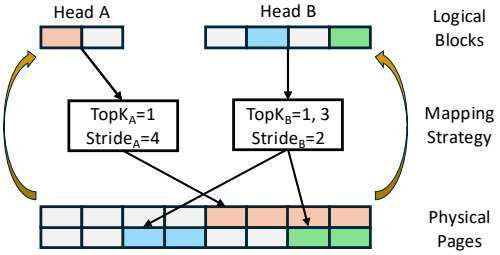


Figure 8: Illustration of the page mapping process. Logical blocks of varying sizes are mapped to contiguous physical pages via a block-to-page stride, enabling variable block size to interface with standard paged KV cache management.

the other, looping over heads or gathering selected blocks into contiguous buffers (Figure 13); three kernels avoid both.

Kernel 1: Fused query–centroid estimation. All centroids of a layer are stored in one flattened array, and a prefix-sum offset array $\text{offset}_{h+1} = \text{offset}_h + N_h$ delimits the segment $[\text{offset}_h, \text{offset}_{h+1})$ of head h. Each thread block reads its head's segment bounds and processes a fixed-size tile, so heads with different N_h are batched into one launch with fully vectorized execution and no padding. INT4 dequantization is fused into the score computation, so the centroids cross the memory bus once, in their compressed form.

Kernel 2: Batched Top-K selection. Given the scores, this kernel selects the Top-K_h blocks of head h under a common token budget T, with $K_h = \lceil T/B_h \rceil$ varying inversely with the block size. When T is not a multiple of B_h, the last selected block is truncated so that every head attends to exactly T tokens; the accuracy gains of *AB-Sparse* therefore come from better block selection rather than from larger coverage. The prefix-sum offsets of Kernel 1 partition the scores by head, so selection is batched as well.

Kernel 3: Heterogeneous paged attention. Physical pages are the finest block size in the candidate set, and every candidate block size is an integer multiple of it, so a block of head h always covers B_h/P consecutive pages. As Figure 8 illustrates, the selected blocks of each head are expanded into page indices through this per-head stride, producing exactly the page table that a standard paged attention kernel consumes. No KV data moves, and the attention kernel itself is unmodified, which is what makes *AB-Sparse* a drop-in replacement in existing serving systems.

4 Evaluation

In this section, we perform quantitative experiments to demonstrate that *AB-Sparse* improves accuracy over existing block sparse attention baselines while preserving throughput. We present accuracy results in §4.2, efficiency results in §4.3, micro study in §4.4, and ablation studies in §4.5.

4.1 Experimental Setup

Hardware and models. We conduct throughput experiments on two hardware platforms: NVIDIA A100-80GB and NVIDIA H800-80GB GPUs. We evaluate *AB-Sparse* on three representative open-source LLMs: Llama-3.1-8B Meta (2024), Qwen3-8B Qwen (2025a), and Qwen3-32B Qwen (2025b), spanning two architecture families and natively supporting context lengths up to 128K tokens.

Benchmarks. We employ two complementary benchmarks for accuracy evaluation: RULER Hsieh et al. (2024) and LongBench Bai et al. (2024). RULER is a synthetic benchmark designed to systematically probe long-context capabilities. It encompasses four task categories: retrieval, multi-hop reasoning, aggregation, and question answering, covering 13 tasks in total. We evaluate at

context lengths from 16K to 96K to assess performance scaling with sequence length. LongBench provides a more realistic evaluation suite comprising real-world long-document understanding tasks across six diverse categories: single-document QA, multi-document QA, summarization, few-shot learning, synthetic tasks, and code completion. This benchmark complements RULER by evaluating *AB-Sparse* on natural text with practical downstream tasks.

Baselines. We implement *AB-Sparse* with 9K lines of Python and CUDA code. We compare *AB-Sparse* against full attention Dao et al. (2022) and two state-of-the-art block sparse attention methods: Quest Tang et al. (2024) and ArkVale Chen et al. (2024a). Quest estimates block importance using per-block min-max pooling centroids, while ArkVale employs bounding-volume centroids for tighter block representation. *AB-Sparse* is applied on top of Quest and ArkVale as a drop-in replacement for their uniform block size assignment, with centroid quantization enabled. For all sparse methods, we fix the KV budget at 4% and the average block size at 32, following the settings adopted in common practice Liu et al. (2025); Wu et al. (2026).

4.2 Accuracy Evaluation

Table 1 reports accuracy results on RULER (left) and LongBench (right). *AB-Sparse* consistently outperforms both baselines across all models and benchmarks. *AB-Sparse*-Quest improves over Quest by 3.51%/2.19%/5.43% on RULER and 2.47%/2.61%/2.62% on LongBench for Llama-3.1-8B/Qwen3-8B/Qwen3-32B, respectively. *AB-Sparse*-ArkVale achieves similar gains, surpassing ArkVale by 3.80%/2.17%/3.16% on RULER and 1.92%/2.19%/1.52% on LongBench. These results confirm that adaptive block size allocation recovers a substantial fraction of the accuracy gap between sparse and full attention. Moreover, AB-Sparse improves accuracy on the vast majority of tasks and context lengths, demonstrating that offline calibration generalizes well. Notably, AB-Sparse consistently improves over both Quest and ArkVale, despite their fundamentally different block representation strategies. This suggests that adaptive block size allocation is agnostic to the underlying block representation, offering a general enhancement for block sparse attention methods.

4.3 Efficiency Evaluation

We evaluate the decoding efficiency of *AB-Sparse* against baselines across context lengths from 64K to 256K tokens. Since ArkVale differs from Quest only in block representation method, their latency characteristics are largely identical. We therefore exclude ArkVale from the efficiency comparison and use Quest as a representative block sparse attention baseline. Figure 9 presents the decoding attention latency of all methods on A100 and H800 GPUs across all three models. *AB-Sparse* matches Quest in latency at shorter contexts and becomes increasingly faster as context length grows. This is because INT4 centroid quantization reduces memory traffic during the estimation stage, an advantage that scales with context length.

Table 1: Accuracy (%) comparison on RULER (left) and LongBench (right) across three models. *AB-Sparse* consistently outperforms baselines across all benchmarks.

	Methods	16K	32K	64K	96K	Avg.	SQA	MQA	SUM	FL	ST	CC	Avg.
Llama-3.1-8B	Full Attention	94.15	92.30	86.41	81.99	88.71	23.15	19.79	25.25	62.31	61.46	58.19	41.69
	Quest	81.57	81.02	77.48	72.83	78.23	20.35	17.93	23.40	58.64	55.59	48.69	37.43
	AB-Sparse-Quest	85.04	85.56	81.49	74.87	81.74	22.19	18.03	25.04	60.85	60.24	53.05	39.90
	ArkVale	82.77	83.50	80.69	75.43	80.60	22.40	18.11	23.57	59.21	56.88	53.18	38.89
	AB-Sparse-ArkVale	88.98	88.43	82.84	77.34	84.40	22.02	19.73	24.87	62.50	59.86	55.87	40.81
Qwen3-8B	Full Attention	91.49	91.19	74.44	71.74	82.22	15.64	13.30	21.78	62.30	65.11	67.33	40.91
	Quest	81.16	82.28	66.88	66.49	74.20	11.97	10.88	20.29	56.45	56.67	52.77	34.84
	AB-Sparse-Quest	84.00	84.01	69.50	68.03	76.39	13.68	12.13	20.86	59.09	61.30	57.61	37.45
	ArkVale	84.51	84.52	68.99	67.90	76.48	13.35	11.93	20.34	57.18	57.26	58.37	36.41
	AB-Sparse-ArkVale	87.29	87.38	70.62	69.29	78.65	14.74	12.44	21.29	59.84	61.20	62.08	38.60
Qwen3-32B	Full Attention	86.25	87.75	84.58	79.47	84.51	18.61	16.64	21.75	54.03	54.58	18.31	30.65
	Quest	77.78	80.09	75.02	72.07	76.24	14.47	15.39	20.56	49.23	47.60	15.50	27.12
	AB-Sparse-Quest	86.98	84.86	79.34	75.50	81.67	16.78	15.87	20.72	52.36	56.27	16.43	29.74
	ArkVale	84.84	84.79	79.19	74.25	80.77	15.56	15.60	20.19	49.89	52.79	15.46	28.25
	AB-Sparse-ArkVale	89.88	86.37	81.19	78.27	83.93	16.41	16.89	21.47	54.68	51.98	17.17	29.77

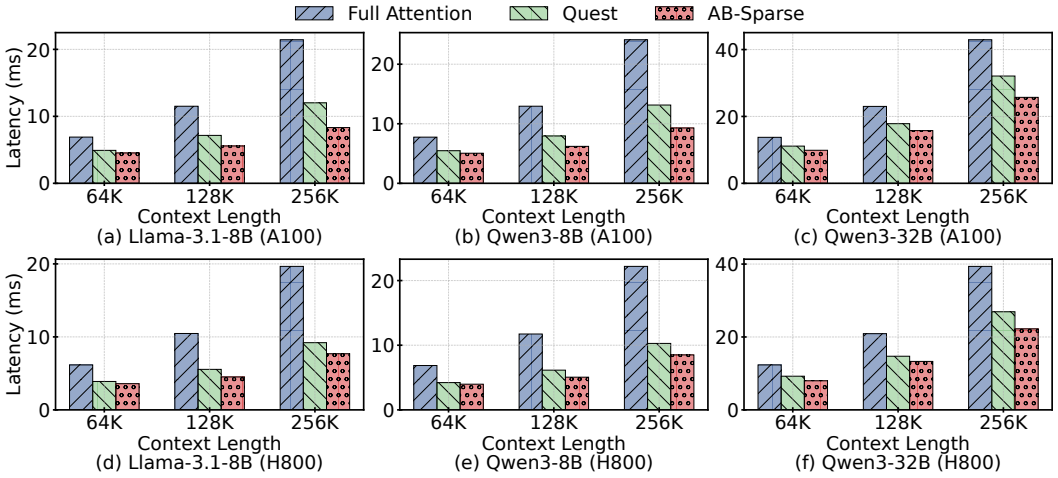


Figure 9: Decoding attention latency (ms) across three models with varying context lengths on A100 and H800 GPUs. *AB-Sparse* achieves increasingly lower latency as context length grows.

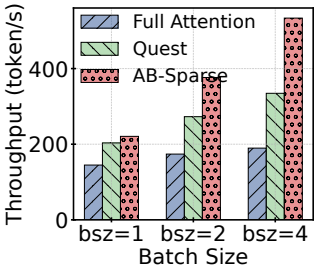


Figure 10: Throughput (tokens/s) with 64K context length and varying batch sizes on Llama-3.1-8B.

	Quest	*AB-Sparse*
AIME24	20.0	23.3
AMC23	47.5	60.0
MATH500	74.0	76.0
Avg.	47.2	53.1

Table 2: Long generation accuracy (%) (pass@4) of Qwen3-8B on three common used reasoning benchmarks with 32K max generation length.

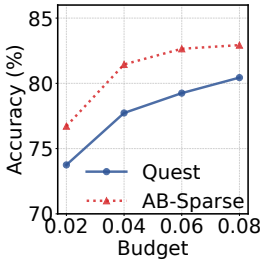


Figure 11: RULER accuracy (%) at 64K context length with varying token budget on Llama-3.1-8B.

4.4 Microscopic Study

Throughput vs. batch size. Figure 10 reports throughput of Llama-3.1-8B on A100 with 64K context length and batch sizes of $\{1, 2, 4\}$. At batch size 1, *AB-Sparse* achieves throughput comparable to Quest; at batch size 4, it reaches 1.59× the throughput of Quest. This improvement stems from two factors: INT4 centroid quantization reduces memory traffic during the estimation stage, and the prefix-sum indexing enables padding-free batched execution across heads with heterogeneous centroid counts, allowing *AB-Sparse* to scale more efficiently as batch size increases.

Long generation accuracy. We additionally evaluate *AB-Sparse* on long-generation tasks using Qwen3-8B Qwen (2025a) on three reasoning benchmarks: AIME24 of Problem Solving (2024), AMC23 of Problem Solving (2023), and MATH-500 Lightman et al. (2023), which feature short inputs with long outputs. We adopt the sampling parameters recommended by Qwen3-8B Qwen (2025a) (top_k = 20, top_p = 0.95, and temperature=0.6) and set the maximum generation length to 32K following DeepSeek-R1 Guo et al. (2025). We sample each input four times and report pass@4 as the accuracy metric. As shown in Table 2, *AB-Sparse*-Quest outperforms Quest across three benchmarks, improving the average pass@4 from 47.2% to 53.1%. This demonstrates that adaptive block size allocation is effective not only for long-input tasks but also for long-generation tasks.

Dynamic token budget. We evaluate Llama-3.1-8B on RULER at 64K context length, varying the token budget ratio from 2% to 8%. As shown in Figure 11, *AB-Sparse* consistently outperforms

Quest across all budget levels by 2.97–3.69%. The persistent gap as the budget increases suggests that adaptive block size allocation provides benefits complementary to simply enlarging the token budget. Additional results on Qwen3-8B are provided in §A.1.

4.5 Ablation Study

Effect of centroid quantization. Figure 12 reports RULER accuracy under different centroid precisions across two models, with BF16 as the unquantized baseline. INT4 quantization achieves accuracy comparable to BF16, confirming that per-channel asymmetric quantization preserves block ranking with negligible accuracy loss. Additional results on Qwen3-32B are provided in §A.2.

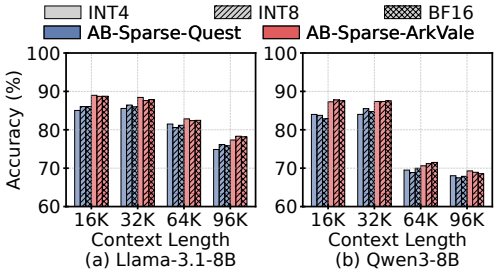


Figure 12: RULER accuracy (%) under different centroid precisions across two models. INT4 quantization achieves accuracy comparable to the unquantized BF16 baseline.

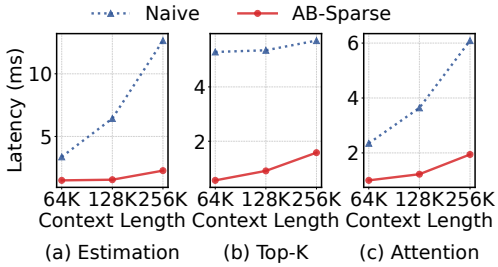


Figure 13: Kernel latency (ms) comparison between the naive implementation and *AB-Sparse* with varying context length. AB-Sparse achieves consistently lower latency.

Effect of custom kernels. Figure 13 compares the latency of the three core operations between the naive implementation and *AB-Sparse*'s custom kernels across context lengths from 64K to 256K. The naive estimation and Top-K kernels loop over heads sequentially due to varying centroid counts, while the naive attention kernel gathers selected KV blocks into contiguous memory before computation. Our kernels consistently achieve lower latency, with speedups of up to 5.6×/9.4×/3.1× for estimation/Top-K/attention, respectively.

5 Related Work

Head-adaptive sparse attention. Existing head-adaptive methods exploit attention-head heterogeneity in different ways: FastGen Ge et al. (2024) profiles attention patterns to select head-specific KV cache policies; Ada-KV Feng et al. (2025) reallocates cache budgets across heads to reduce eviction-induced errors; TCA-Attention You et al. (2025) calibrates head-specific sparsity budgets offline and selects informative tokens online; MInference Jiang et al. (2024) and DuoAttention Xiao et al. (2024b) profile per-head sparsity patterns. AB-Sparse instead adapts block size, targeting selection granularity, and is orthogonal to budget allocation or sparsity pattern.

KV cache quantization. KIVI Liu et al. (2024d) applies different quantization strategy for key and value vectors, while KVQuant Hooper et al. (2024) combines pre-RoPE quantization with outlier handling. In contrast, AB-Sparse quantizes block centroids used solely for importance estimation and ranking, while preserving the original precision of selected KV vectors.

6 Conclusion

Block sparse attention loses accuracy because it summarizes every head at the same granularity although heads differ widely in the granularity they need. *AB-Sparse* gives each head its own block size, chosen once by calibration and made free at run time by INT4 centroids and kernels that batch heterogeneous heads on an unmodified paged KV cache, lifting Quest and ArkVale by 3.0–3.7 points on RULER and 1.9–2.6 points on LongBench with comparable or higher decoding throughput.

AI USE STATEMENT

In this work, we used generative AI tools solely to polish the manuscript to improve readability and language quality. We have not used generative AI tools for research ideation, methodology design, experiment implementation, data analysis, or result interpretation. We have reviewed all AI-assisted edits and verified that the meaning and technical content of the original text were preserved. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, 2023.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 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, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. Arkvale: Efficient generative llm inference with recallable key-value eviction. *Advances in Neural Information Processing Systems*, 37:113134–113155, 2024a.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Jing Liu, et al. Retroinfer: A vector-storage approach for scalable long-context llm inference. *arXiv preprint arXiv:2505.02922*, 2025.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, et al. Magicpig: Lsh sampling for efficient llm generation. *arXiv preprint arXiv:2410.16179*, 2024b.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-kv: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for llms. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 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.

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.

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

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. Infinigen: Efficient generative inference of large language models with dynamic kv cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172, 2024.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

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

Di Liu, Yifei Liu, Chen Chen, Zhibin Yu, Xiaoyi Fan, Quan Chen, and Minyi Guo. S-HPLB: efficient LLM attention serving via sparsity-aware head parallelism load balance. *CoRR*, abs/2603.10353, 2026. doi: 10.48550/ARXIV.2603.10353. URL https://doi.org/10.48550/arXiv.2603.10353.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. *arXiv preprint arXiv:2412.03213*, 2024c.

Guangda Liu, Chengwei Li, Zhenyu Ning, Jing Lin, Yiwu Yao, Danning Ke, Minyi Guo, and Jieru Zhao. Freekv: Boosting kv cache retrieval for efficient llm inference. *arXiv preprint arXiv:2505.13109*, 2025.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344, 2024d.

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.

Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, et al. Repoagent: An llm-powered open-source framework for repository-level code documentation generation. *arXiv preprint arXiv:2402.16667*, 2024.

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

Art of Problem Solving. Amc problems and solutions. https://artofproblemsolving.com/wiki/index.php?title=AMC_Problems_and_Solutions, 2023.

Art of Problem Solving. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions, 2024.

Qwen. Qwen3-8b. https://huggingface.co/Qwen/Qwen3-8B, 2025a.

Qwen. Qwen3-32b. https://huggingface.co/Qwen/Qwen3-32B, 2025b.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

wikipedia. wikipedia. https://huggingface.co/datasets/wikimedia/wikipedia, 2025.

Fang Wu, Congming Gao, Weixi Zhu, and Jiwu Shu. PRKV:page restruct KV cache for high accuracy and efficiency LLM generation, 2026. URL https://openreview.net/forum?id=7FM0GBFhe5.

Wenhao Wu, Yizhong Wang, Guangxuan Xiao, Hao Peng, and Yao Fu. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024.

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, 2024a.

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. *arXiv preprint arXiv:2410.10819*, 2024b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Zukang Yang, Zixuan Zhu, and Jennifer Zhu. Curiousllm: Elevating multi-document question answering with llm-enhanced knowledge graph reasoning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 274–286, 2025b.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Zeng You, Yaofo Chen, Shuhai Zhang, Zhijie Qiu, Tingyu Wu, Yingjian Li, Yaowei Wang, and Mingkui Tan. Training-free context-adaptive attention for efficient long context modeling. *CoRR*, abs/2512.09238, 2025. doi: 10.48550/ARXIV.2512.09238. URL https://doi.org/10.48550/arXiv.2512.09238.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Appendix

A.1 Additional Token Budget Results

Figure 14 reports accuracy on Qwen3-8B evaluated with RULER at 64K context length under varying token budget ratios. Consistent with the Llama-3.1-8B results in §4.4, *AB-Sparse* outperforms Quest across all budget levels, with improvements ranging from 1.16% to 2.81%. The trend confirms that the benefits of adaptive block size allocation generalize across model architectures and are complementary to increasing the token budget.

A.2 Additional Centroid Quantization Results

Figure 15 presents the centroid quantization ablation on Qwen3-32B, complementing the Llama-3.1-8B and Qwen3-8B results in §4.5. The same trend holds: INT4 centroid quantization maintain accuracy on par with the BF16 baseline across all context lengths, further validating the robustness of per-channel asymmetric quantization across model scales.

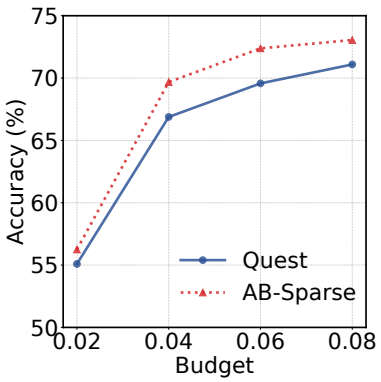


Figure 14: RULER accuracy (%) at 64K context length with varying token budget on Qwen3-8B. *AB-Sparse* consistently outperforms the baseline.

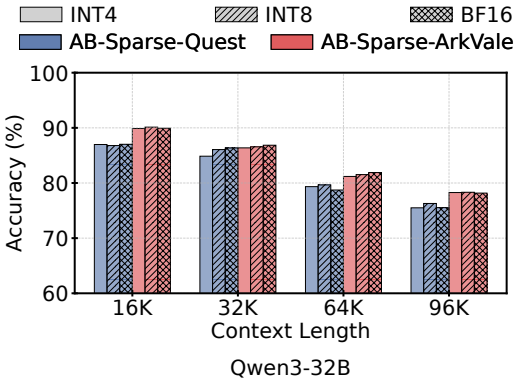


Figure 15: RULER accuracy (%) under different centroid precisions on Qwen3-32B. INT4 quantization achieves accuracy comparable to the unquantized BF16 baseline.

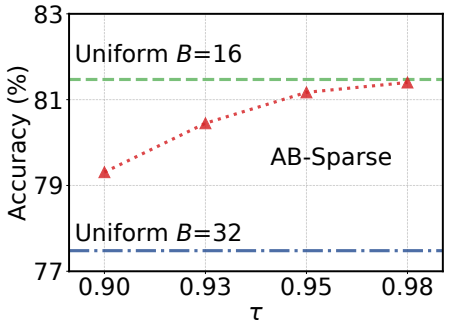


Figure 16: RULER accuracy (%) as τ sweeps $\{0.90, 0.93, 0.95, 0.98\}$; all configurations use no quantization centroids.

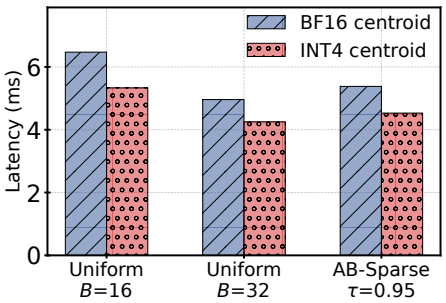


Figure 17: Comparison of decoding latency (ms) between *AB-Sparse* and baseline methods, with and without quantization.

A.3 Additional Ablation Study Results

We compare AB-Sparse against uniform B=16 and uniform B=32 on Llama-3.1-8B, measuring accuracy on RULER and decode attention latency at 64K context on one A100. The results are presented in Figure 16 and Figure 17. The threshold τ is the single knob of the calibration: sweeping

it over $\{0.90, 0.93, 0.95, 0.98\}$ moves the accuracy from 79.31 to 81.40, spanning the interval between uniform B=32 and uniform B=16. What makes the fine end of that sweep affordable is our observation that centroids only rank blocks and never enter the attention output, so they can be quantized to 4 bits without touching output fidelity: INT4 centroids cut centroid bytes fourfold and reduce decode attention latency by 14–18% in every configuration, from 6.47 to 5.33 ms at uniform B=16 and from 5.38 to 4.53 ms for AB-Sparse.

A.4 LIMITATIONS

AB-Sparse currently selects block sizes from a fixed candidate set $\{16, 32, 64\}$. Exploring a broader block size space may further improve the accuracy-efficiency trade-off. Additionally, the calibration-driven profiling uses a subset of samples from Wikipedia, a generic and domain-agnostic corpus. Our evaluation across two benchmarks covering diverse long-context scenarios demonstrates that the generalization. We leave the investigation on comprehensive calibration selection for future work.