NOT ALL TOKENS DESERVE EQUAL RESOLUTION: SCORE-ADAPTIVE INDEXING FOR SPARSE ATTENTION IN LONG-CONTEXT LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Indexer-based sparse attention reduces the cost of the main attention operator by retrieving only a small subset of the context, but the indexing stage itself is expensive for long sequence lengths. Hierarchical indexing reduces this cost by representing contiguous regions with compact summaries and refining only selected candidates. However, fixed-size partitions assume that every region can be represented equally well by one summary. While mean pooling is reliable when nearby keys are similar, it can obscure important token-level variation when they are not. We introduce SCALE (Score-Adaptive Indexing for Long-Context LLMs), a training-free hierarchical index that improves summary fidelity under a fixed budget by adapting which tokens are grouped together. For a fixed chunk, the mean key minimizes squared reconstruction error. For the DSA scoring function used in our evaluation, we further show that key dispersion controls an upper bound on the score approximation error induced by mean pooling. SCALE therefore splits high-dispersion regions into more chunks and merges homogeneous neighbors, while retaining the original mean representation and token-level indexer for exact refinement. Using the same number of chunks, SCALE improves Top-2,048 candidate recall over fixed-block indexing by 0.32–2.37 percentage points while consistently reducing score approximation error. With one summary per 64 tokens on average, SCALE-64 improves pooled RULER accuracy from 65.3 to 82.1 and LongBench-v2 accuracy from 46.4 to 50.4 over fixed-block HISA-64, a recent strong fixed-block hierarchical baseline. On the 128K serving workload, the same configuration reduces end-to-end latency by 36% relative to flat DSA (DeepSeek Sparse Attention), from 65.30 s to 41.69 s, while matching DSA on LongBench-v2 (50.4) and remaining close on RULER (82.1 versus 84.7).

1 INTRODUCTION

Long-context LLMs increasingly rely on sparse attention to avoid attending to the entire history at every layer. Many recent sparse-attention designs first use a lightweight indexer to select a small subset of relevant context, then apply the more expensive attention operator only to the selected tokens. DSA (Liu et al., 2025) performs fine-grained token-level retrieval. QSA (Qiu et al., 2026) reduces indexing cost by scoring fixed-size compressed micro-blocks. HISA (Xu et al., 2026) and LongCat-HI (Zan et al., 2026) use coarse-to-fine retrieval over fixed blocks or pages. These systems reduce attention work, but share a hidden assumption: every region of the sequence is equally easy to summarize.

Uniform partitioning ignores that regions differ in how well they can be summarized. When nearby index keys are similar, a mean key can represent many tokens; when they vary, averaging may obscure important token-level structure and cause relevant regions to be discarded before refinement. We refer to this chunk granularity as *indexing resolution*: smaller chunks provide finer resolution, while larger chunks provide coarser resolution. Increasing the number of summaries would provide fine resolution everywhere, but it would also lengthen the coarse scan and raise indexing cost.

Under a fixed summary budget, resolution must therefore be allocated selectively. This leads to a simple question: *under a fixed number of summaries, where should indexing resolution be spent?*

Figure 1 shows a toy instance of this allocation. Our answer starts from the summary itself. We use one mean key per chunk as a practical balance between representation quality and serving cost.

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.

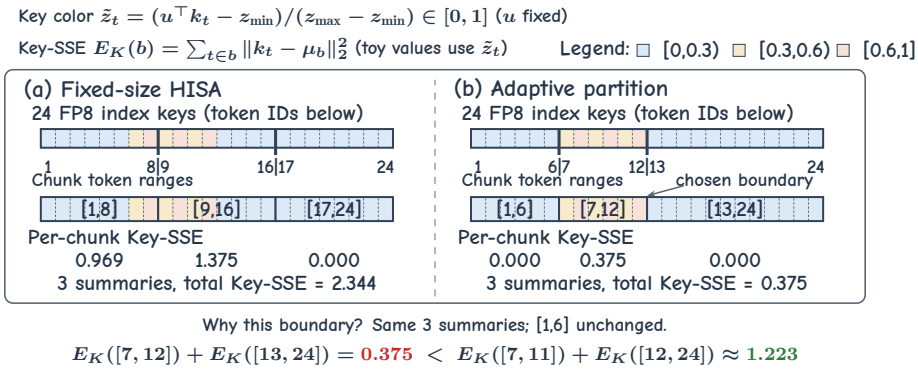


Figure 1: Adaptive boundaries improve mean-summary fidelity at the same three-summary budget. Labels $[a, b]$ are inclusive token ranges sharing one mean summary. For these toy keys, minimizing total Key-SSE over three contiguous chunks gives cuts after tokens 6 and 12, reducing Key-SSE from 2.344 (fixed length-8 chunks) to 0.375.

With one stored vector per chunk, mean pooling matches HISA's storage and scoring cost while minimizing squared reconstruction error. SCALE therefore keeps mean pooling and optimizes which tokens share each mean. We measure how reliably a mean represents its chunk using Key-SSE, the sum of squared deviations of the stored index keys from their mean. Low Key-SSE indicates that many tokens can share one summary; high Key-SSE indicates that finer grouping may be needed. For the DSA scoring function used in our evaluation, we show that key dispersion controls an upper bound on the score approximation error introduced by mean pooling (Section 3.1). Because this statistic depends only on stored keys, it can be computed before future queries arrive. The bound is a partitioning surrogate, not a Top-K guarantee.

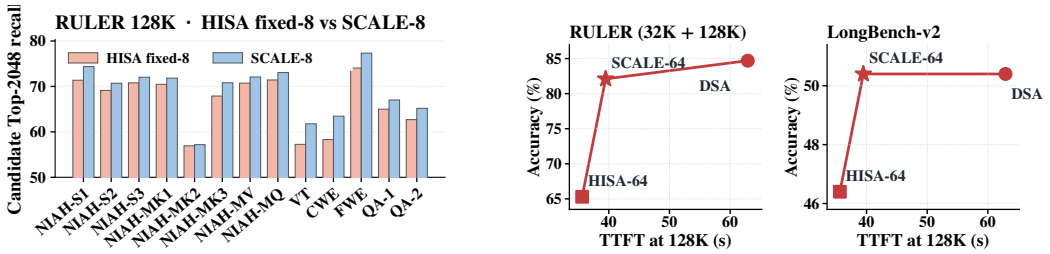
We therefore introduce SCALE, a training-free hierarchical index with variable-length chunks while keeping the number of summaries fixed. SCALE first splits high-dispersion regions into smaller chunks. It then merges neighboring chunks whose shared mean adds the least squared error, following Ward's criterion (Ward Jr, 1963), until it reaches the target number of summaries. The resulting intervals have variable lengths but retain the same mean-key representation as fixed-block indexing. At query time, SCALE scores these summaries, expands the selected chunks up to a fixed token budget, and rescores their tokens with the original indexer. Thus, SCALE changes neither model weights nor the sparse-attention interface; it changes only which tokens share a summary.

Across matched $N/8$, $N/32$, $N/64$, and $N/128$ summary budgets on 128K RULER traces, adaptive partitioning improves Top-2,048 candidate recall over fixed blocks by 0.32–2.37 percentage points under the same 8,192-token refinement budget, while consistently reducing score approximation error. At the matched $N/64$ budget, SCALE-64 improves pooled RULER accuracy from 65.3 to 82.1 and LongBench-v2 accuracy from 46.4 to 50.4 over HISA-64. Serving speed stays close to HISA-64, with a lower decode TPOT (17.46 versus 17.56 milliseconds). Relative to flat DSA, the same configuration matches LongBench-v2 (50.4) and remains close on RULER (82.1 versus 84.7), while reducing 128K end-to-end latency from 65.30 to 41.69 seconds. Figure 2 summarizes the matched-budget retrieval gain (Figure 2a) and the downstream trade-off between quality and latency (Figure 2b).

We address a limitation of fixed-size hierarchical sparse-attention indexing: under a fixed summary budget, different regions of a long context may require different indexing resolutions to be represented faithfully. Our contributions are the following:

1. A query-independent criterion for deciding where finer indexing is needed: we use Key-SSE to measure how faithfully a chunk is represented by its mean key, and show for the DSA scoring function that this quantity controls an upper bound on the score error introduced by mean pooling.

2. A fixed-budget indexing algorithm for turning this criterion into adaptive chunk boundaries: SCALE allocates more boundaries to high-Key-SSE regions and removes redundant boundaries using Ward merging until the target number of summaries is reached. The resulting chunks are then selected under a fixed token budget and exactly rescored by the original token-level indexer.



(a) Matched-summary retrieval fidelity.

(b) Downstream quality versus TTFT.

Figure 2: Matched-budget retrieval fidelity and downstream quality versus TTFT at 128K. Left: adaptive $N/8$ boundaries improve recall over fixed length-8 chunks. Right: each marker pairs quality and latency from the same configuration; upper left is better.

3. State of the art performance: We implement and evaluate SCALE end to end with incrementally updated Key-SSE trees and CUDA-graph-compatible GPU kernels for partitioning and candidate selection. Across $N/8$–$N/128$ budgets, candidate recall improves by 0.32–2.37 percentage points over fixed blocks. At $N/64$, SCALE-64 raises pooled RULER accuracy from 65.3 to 82.1 over HISA-64, matches DSA on LongBench-v2 (50.4), and reduces end-to-end latency by 36% relative to DSA.

2 Background

2.1 Indexer Setup

DeepSeek-V3.2 is a mixture-of-experts long-context LLM equipped with DeepSeek Sparse Attention (DSA) (Liu et al., 2025). DSA uses a lightweight auxiliary indexer to rank historical keys before main attention and associates every query token with $H = 64$ indexer heads. Let $q_{i,h} \in \mathbb{R}^D$ be head h of query token i, $k_t \in \mathbb{R}^D$ the shared index key at historical position t, and $w_{i,h} \in \mathbb{R}$ the learned head weight. The production indexer score is $I(i,t) = \sum_{h=1}^{H} w_{i,h}\,\mathrm{ReLU}(q_{i,h}^\top k_t)$ (1). DSA evaluates Eq. (1) for every visible key and returns its Top-K, with $D = 128$ and $K = 2{,}048$ in DeepSeek-V3.2. This flat scan is exact but linear in the prefix length for every query.

HISA (Xu et al., 2026) uses the fixed-block hierarchy shown in Figure 1(a). It mean-pools one key per block, ranks the pooled summaries, expands the highest-ranked blocks to a token budget C, and evaluates Eq. (1) exactly inside that candidate set before returning Top-K. Figure 1(b) shows how SCALE keeps this coarse-to-fine interface but replaces fixed blocks with variable-length chunks. Its partition is built once from index keys, independently of the current query, and is reused by subsequent queries. SCALE preserves the exact DSA score during token-level refinement; candidate budgets and exact sink/recent guards are specified in Section 4.1.

3 Method

3.1 Key-SSE objective and score-error bound

This subsection develops the query-independent signal that SCALE uses to place chunk boundaries. We first measure how well one mean represents a span, then derive a shared split–merge cost, and finally connect that key-space objective to DSA score error. For a contiguous interval b, define its mean μ_b and key-space sum of squared errors (Key-SSE) $E_K(b)$ by

$$\mu_b = \frac{1}{|b|}\sum_{t\in b} k_t, \qquad E_K(b) = \sum_{t\in b}\|k_t - \mu_b\|_2^2 = \sum_{t\in b}\|k_t\|_2^2 - |b|\|\mu_b\|_2^2. \tag{2}$$

Key-SSE measures the total squared deviation of the stored index keys from the chunk mean. A smaller value means that the mean represents the chunk more faithfully. We use this quantity as the P-key partitioning criterion. For fixed b, μ_b is squared-error-optimal among one-vector key summaries, but not necessarily for Top-K retrieval. Unlike a query-conditioned score field, E_K can be constructed while the index-key cache is written and remains valid for every future query; its

sufficient statistics are the count, vector sum, and squared norm. We divide the indexed prefix into complete 256-token spans and build an independent dyadic tree for each span. A tree's root node represents all 256 tokens in the span, while its leaves represent individual tokens. This tree root is distinct from the 8,192-token SGLang prefill chunk used by the serving pipeline; each complete prefill chunk contains 32 such roots. The residual suffix shorter than one root is kept in the exact local region. Adaptive chunks do not cross root boundaries, so extending the indexed prefix creates only new trees: existing nodes are reused bit-for-bit, and only the new roots are built.

To decide which adjacent chunks to merge, SCALE uses the Ward cost as its merge criterion. For a pair of adjacent chunks, this cost is exactly the increase in Key-SSE caused by replacing their two means with a joint mean (Ward Jr, 1963).

$$\Delta_K(a,b) = \frac{|a||b|}{|a|+|b|}\|\mu_a - \mu_b\|_2^2 \geq 0. \tag{3}$$

The same quantity is the Key-SSE reduction obtained by splitting a parent into its children. Using it for both operations aligns the two construction stages: splitting adds boundaries with large Key-SSE reductions, while merging removes boundaries with the smallest Ward costs.

Score-error bound. The role of Key-SSE in SCALE is to identify chunks whose mean is likely to distort the DSA indexer score. The following proposition formalizes this connection.

Proposition 1 (Lipschitz bound for DSA mean pooling)**.** *For query position i, let $A_i = \sum_{h=1}^{H} |w_{i,h}|\|q_{i,h}\|_2$. For every chunk b, the root-mean-square score error satisfies*

$$\left(\frac{1}{|b|}\sum_{t\in b} |I(i,t) - I(i,\mu_b)|^2\right)^{1/2} \leq A_i\sqrt{\frac{E_K(b)}{|b|}}. \tag{4}$$

For any partition P of the N historical tokens, the global squared error satisfies

$$\frac{1}{N}\sum_{b\in P}\sum_{t\in b} |I(i,t) - I(i,\mu_b)|^2 \leq \frac{A_i^2}{N}\sum_{b\in P} E_K(b). \tag{5}$$

Proposition 1 motivates SCALE's partition objective. Both construction stages act on the quantity on the right-hand side of Eq. (5): splitting favors boundaries that reduce E_K, while Ward merging removes the adjacent boundary with the smallest increase in E_K. Among partitions with the same summary budget, a lower total Key-SSE therefore lowers the right-hand side of the bound for every query; only the multiplicative factor A_i depends on the query. Appendix B gives the proof and extends the result to the requantized FP8 summaries used by the implementation.

3.2 Fixed-budget adaptive partitioning

Figure 3 shows the two stages of fixed-budget adaptive partitioning: (a) *Fixed-budget Key-SSE split* creates M_0 candidate intervals, and (b) *Target-count Ward merge* reduces them to T summaries. The reference budgets are $M_0 = N/8$ leaves and $T = N/64$ summaries. Section 4.1 defines the Sx/My names for these serving configurations.

Fixed-budget Key-SSE split. Figure 3(a) combines a search for the leaf penalty λ, a Key-SSE dynamic program, and exact-count repair. The dynamic program allocates candidate boundaries to regions where they most reduce Key-SSE. For a tree node b with children ℓ and r, the penalized dynamic program is

$$F_b(\lambda) = \min\{E_K(b) + \lambda,\ F_\ell(\lambda) + F_r(\lambda)\}. \tag{6}$$

For a fixed λ, Eq. (6) is evaluated bottom-up: the first term keeps b as one interval, and the second splits it recursively. The explicit decision shown above the tree in Figure 3(a) is

$$\text{split } b \quad\Longleftrightarrow\quad \underbrace{F_\ell(\lambda) + F_r(\lambda)}_{\text{split cost}} < \underbrace{E_K(b) + \lambda}_{\text{keep cost}}. \tag{7}$$

Otherwise, including equal costs, keep b. If both children remain unsplit, $F_c(\lambda) = E_K(c) + \lambda$ for $c \in \{\ell, r\}$, and this comparison simplifies to

$$E_K(b) - E_K(\ell) - E_K(r) > \lambda.$$

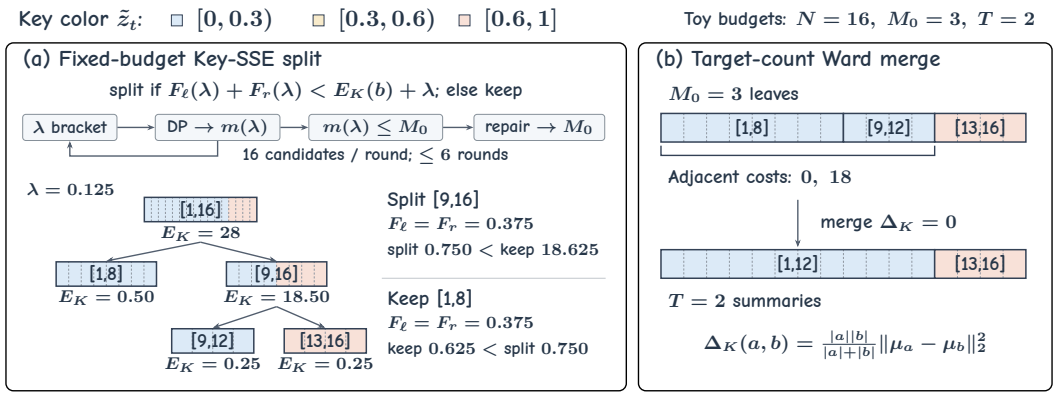


Figure 3: Fixed-budget adaptive partitioning. (a) Fixed-budget Key-SSE split builds M_0 leaves under a split penalty. (b) Target-count Ward merge removes the cheapest boundaries until T summaries remain. Colors denote projected key values.

Thus the Key-SSE reduction must exceed the penalty for one extra interval in this direct two-child case. If either child is further partitioned, Eq. (7) uses its optimized subtree cost instead. The three-level tree in Figure 3(a) labels every interval's E_K. At $\lambda = 0.125$, $[9, 16]$ splits because $0.750 < 18.625$, while $[1, 8]$ is kept because $0.625 < 0.750$. Its three terminal intervals already meet the toy budget $M_0 = 3$, so repair is unnecessary.

The resulting leaf count $m(\lambda)$ is non-increasing in λ. The upper flow in Figure 3(a) searches this penalty by evaluating 16 candidates in parallel per round and narrowing the bracket around the first feasible count $m(\lambda) \le M_0$. After at most six rounds, it selects the feasible endpoint and, if needed, adds the highest-gain available splits until the count is exactly M_0. Appendix A and Algorithm 1 give the bracket initialization, parallel evaluation, repair details, and complete pseudocode.

Target-count Ward merge. Figure 3(b) reduces the M_0 split intervals to T summaries. Each round computes the adjacent Ward costs in Eq. (3), forms a deterministic non-overlapping matching among the cheapest eligible edges, and merges without overshooting T. Costs are recomputed on the new adjacency. In the toy example, the two costs are 0 and 18; merging $[1, 8]$ with $[9, 12]$ removes the zero-cost boundary and reaches $T = 2$. Four non-empty rounds suffice in our full-size traces. Algorithm 1 in Appendix A gives the complete construction.

3.3 Fixed-shape serving path

Figure 4 and the three paragraphs below use the same organization: (a.1) Prefill: data view, (a.2) Prefill: stream loop, and (b) Decode with a fixed prompt partition. Let $B = 8{,}192$ and let $\mathcal{C}_i = [(i-1)B, iB)$ denote the ith complete SGLang prefill chunk ($i \ge 1$), distinct from an adaptive interval. For each layer, P_i contains the partition and summaries of the completed-root prefix through $\mathcal{C}_i$; $\mathcal{C}_{i+1}$ consumes P_i.

(a.1) Prefill: data view. The historical region in Figure 4(a.1) uses P_{i-1} for hierarchical selection. Each query applies the original 64-head score in Eq. (1) to its interval means. Padding intervals receive $-\infty$; sink intervals receive $+\infty$. Intervals are consumed in descending score order until they contribute C historical tokens, clipping the crossing interval to its required prefix. This fixes the fine-stage work despite variable interval lengths; clipping is a work-budget policy, not a within-interval ranking. The current chunk is not summarized: every causally visible current key receives a dense indexer score, while future keys remain masked.

(a.2) Prefill: stream loop. On the main stream, our batched weighted-radix kernel uses interval-length-weighted histograms, breaks ties by interval index, and emits candidate IDs in logical order. The selected set equals sorting by (score descending, interval index ascending) and clipping to C tokens, while avoiding a full per-row sort.

Figure 4(a.2) instantiates this loop for SGLang (Zheng et al., 2024) chunked prefill. The first chunk $\mathcal{C}_1 = [0, B)$ uses flat causal DSA scoring because P_0 is absent, then schedules P_1. Chunk $\mathcal{C}_2 = [B, 2B)$ reuses P_1 for history $[0, B)$ and densely scores its current causal keys in $[B, 2B)$. Here

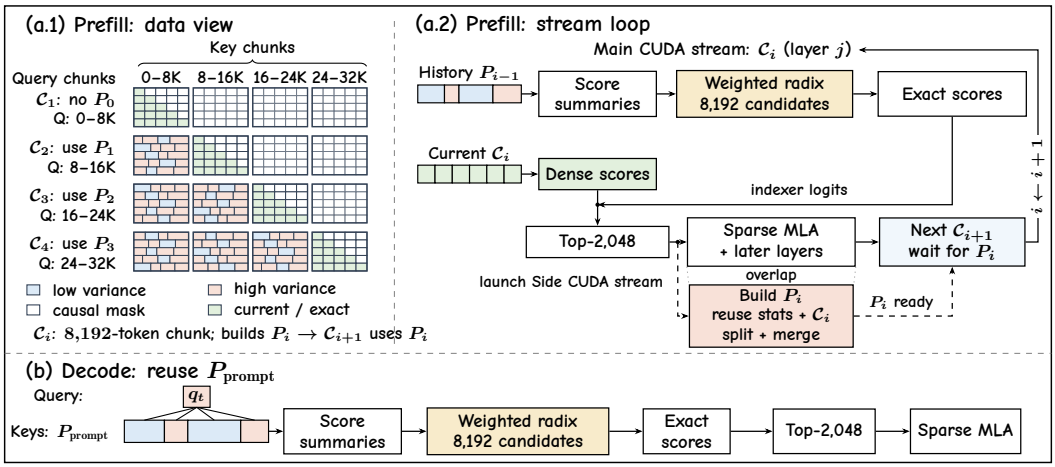


Figure 4: Fixed-shape serving path. (a.1) Prefill: data view: rows are query chunks $\mathcal{C}_i$ and columns are key chunks. (a.2) Prefill: stream loop. (b) Decode with a fixed prompt partition.

history equals the $C = 8{,}192$ candidate budget, so it admits the entire first chunk; longer histories are pruned. In general, selected historical tokens are exactly rescored, while the unindexed tail and current causal chunk receive dense indexer scores. Both sets of indexer logits enter one split-row radix Top-$2{,}048$; MLA then attends to the selected raw tokens. Thus summaries guide historical candidate selection, while dense-causal scoring admits every visible current key to Top-K competition without guaranteeing that it is selected. Immediately after Top-K, the main stream launches the side-stream build of P_i from keys through $\mathcal{C}_i$, overlapping construction with MLA and later layers. The next chunk waits on its ready event before using P_i at the same layer. Thus advancing i makes the newly published partition the next iteration's P_{i-1}.

The side-stream branch reuses historical Key-SSE statistics, extends them with the current $8{,}192$ keys, and then runs two λ rounds without exact repair, two soft Ward rounds, one target-count round, and a bounded device-only fallback. Static capacities and padding preserve prefix coverage and $m \leq T$ under CUDA graphs. We evaluate this fixed-shape approximation separately from the exact-count construction in Figure 3.

The *dense-final* policy retains flat DSA scoring for the final prompt chunk, which determines the first generated token. Appendix A and Algorithm 2 give the full prefill schedule, capacity guarantee, and tensor flow.

(b) Decode with a fixed prompt partition. Figure 4(b) shows this decode path. Decode binds the final prompt partition once; each completed block of 64 generated tokens is appended as one fixed-length summary while the open block stays exact. Every query scores the prompt and decode summaries, reserves sink/tail guards, fills the remaining token budget with the same weighted-radix kernel, and exactly rescans the selected raw tokens before returning Top-$2{,}048$ IDs to sparse MLA. Fixed-capacity workspaces and launch sequences keep the path CUDA-graph safe; Algorithm 3 in Appendix A gives the decode step.

3.4 Complexity and Implementation

We count evaluations of the indexer score in Eq. (1), not MLA or measured kernel time. Let N be the indexed prefix length, T the realized number of summaries, C the token refinement budget, and Q the number of prefill queries. Flat DSA scores all N index keys, costing $O(QNDH)$ in prefill and $O(NDH)$ for one decode query. SCALE instead scores T summaries and exactly rescores the C admitted tokens, giving $O(Q(T + C + W)DH)$ in prefill and $O((T + C)DH)$ in decode, where $W \leq Q + 255$ is the exact causal-local window. Here $T = N/64$ for SCALE-64, while SCALE-128 realizes at most $N/128$ summaries.

Partition construction is separate amortized work. A full Key-SSE tree costs $O(ND)$, while extending it for Q new tokens costs $O(QD)$. Each dynamic-program evaluation and repair is linear in the tree size, and each Ward round costs $O(M_0 D)$. The implementation reuses existing tree statistics, runs

the fixed-shape split–merge schedule on a side stream, and uses weighted radix selection to maintain a fixed token budget for variable-length chunks. Further kernel, workspace, and correctness details are in Appendix C.

4 EXPERIMENTS

4.1 QUESTIONS AND PROTOCOL

We organize the evaluation around three claims. *First*, at a matched summary and candidate budget, does adapting chunk boundaries using key dispersion improve candidate fidelity over fixed blocks? *Second*, what end-to-end trade-off between quality and latency results when the adaptive index is used during prefill and decode? *Third*, what costs and gains arise from constructing and selecting the adaptive partition? Every plotted point pairs downstream quality and latency from the same named configuration.

Unless stated otherwise, all serving experiments use DeepSeek-V3.2 (Liu et al., 2025) in SGLang (Zheng et al., 2024) on one node with $8\times$ NVIDIA H20 GPUs and tensor parallelism 8. The indexer returns $K = 2{,}048$ tokens from a candidate budget $C = 8{,}192$. Each run keeps the first $S = 64$ attention-sink tokens (Xiao et al., 2024) and the most recent $L = 256$ tokens exact. The prompt is processed in chunks of $8{,}192$ tokens. Decode uses CUDA graphs at batch size one. All compared methods receive identical token IDs, model weights, precision, sink/local guards, request order, and random seed. Operator tables report median CUDA-event time after warm-up. We validate each optimized operator against a functionally equivalent reference implementation; Appendix D reports operator latency and speedups. The latest end-to-end table reports the mean of three post-warm-up requests; all individual measurements are retained in the provenance.

Baselines. DSA is the official flat indexer (Liu et al., 2025). **HISA-**64 (Xu et al., 2026) uses fixed 64-token means and selects 128 blocks, matching both SCALE's $N/64$ summary count and its $8{,}192$-token refinement budget. We denote a configuration by Sx/My, where Sx sets the split budget to N/x leaves and My sets the final merge target to N/y summaries. My is that divisor, not the summary count M used elsewhere, so M64 does not mean $M{=}64$. The relaxed path only guarantees that the leaf and summary counts do not exceed these capacities, so x and y are nominal average chunk lengths. **SCALE-**64 is the relaxed S8/M64 construction ($N/8 \to N/64$); **SCALE-**128 is S32/M128 ($N/32 \to N/128$). Both use the relaxed constructor in intermediate prefill chunks and retain flat DSA for the final prompt chunk (the *dense-final* policy). An exact-count $N/32 \to N/128$ construction is included only as an implementation ablation. The relaxed constructor truncates λ search to two rounds, omits exact-count repair, and applies the soft Ward and fixed-capacity schedule in Section 3.3. SCALE-64 and SCALE-128 have paired downstream measurements in Tables 1 and 2.

4.2 QUALITY AND CANDIDATE FIDELITY

Free-generation quality uses the same strict held-out prompts for every arm. RULER (Hsieh et al., 2024) contains 13 subtasks at 32K and 128K with 14 prompts per task/length. LongBench-v2 (Bai et al., 2025) contains 503 questions; our paired held-out evaluation uses 401 multiple-choice prompts common to every arm, grouped by context length and difficulty. Generation is greedy and evaluators are shared across arms. We interpret each sparse-prefill point directly against DSA and HISA rather than transferring quality from a different prefill policy.

For diagnostic traces we additionally report Top-$2{,}048$ recall, prefix recall, and positive-logit mass recall against flat DSA. These metrics explain a miss but do not replace downstream generation; prior work likewise shows that Top-K attention approximations can degrade task quality (Chen et al., 2025). In particular, isolated needles can be diluted by any mean-summary method, including fixed HISA.

As a cross-model check, Table 1 also reports GLM-5-FP8 at 128K only. At the matched $N/64$ budget, SCALE-64 raises average accuracy over HISA-64 from 72.9 to 87.5, recovering 68% of the HISA-to-DSA gap. We treat this as quality robustness; matched GLM-5 latency was not measured. Quality and latency for SCALE-64 and SCALE-128 are paired across Tables 1, 2, and 10: the same dense-final configurations, measured on separate workload runs.

Table 1: RULER accuracy (%) on held-out prompts, grouped by model and context range. SCALE-64 and SCALE-128 use final summary budgets $N/64$ and $N/128$, respectively; each comparison uses the same 8,192-token refinement budget as its HISA baseline.

Task	DeepSeek-V3.2: RULER 32K+128K					GLM-5-FP8: RULER 128K		
	DSA	HISA-128	HISA-64	SCALE-128	SCALE-64	DSA	HISA-64	SCALE-64
Single-Needle 1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
Single-Needle 2	96.4	64.3	71.4	85.7	96.4	100.0	85.7	100.0
Single-Needle 3	100.0	64.3	85.7	96.4	100.0	100.0	100.0	100.0
Multi-Key 1	89.3	46.4	75.0	78.6	82.1	100.0	92.9	100.0
Multi-Key 2	100.0	21.4	28.6	96.4	100.0	100.0	14.3	92.9
Multi-Key 3	100.0	28.6	35.7	100.0	100.0	100.0	14.3	85.7
Multi-Value	89.3	49.1	61.6	58.9	75.9	100.0	83.9	89.3
Multi-Query	92.0	58.0	66.1	75.9	81.2	100.0	87.5	100.0
Var. Tracking	42.1	50.7	59.3	45.7	45.7	100.0	98.6	100.0
Common Words	77.1	50.7	69.6	52.1	61.4	90.0	44.3	34.3
Freq. Words	96.4	98.8	98.8	98.8	98.8	100.0	97.6	100.0
SQuAD	71.4	57.1	57.1	71.4	78.6	78.6	78.6	78.6
HotpotQA	46.4	39.3	39.3	46.4	46.4	57.1	50.0	57.1
Avg.	**84.7**	**56.1**	**65.3**	**77.4**	**82.1**	**94.3**	**72.9**	**87.5**

Table 2: LongBench-v2 multiple-choice accuracy (%) on the 401-prompt held-out set, split by context length and difficulty. SCALE-64 and SCALE-128 use final summary budgets $N/64$ and $N/128$, respectively, with the same 8,192-token refinement budget.

Method	Short		Medium		Long		Overall
	Easy	Hard	Easy	Hard	Easy	Hard	
DSA	47.6	51.9	60.6	47.3	50.0	41.3	50.4
HISA-128	47.6	50.0	46.5	42.7	50.0	47.8	46.9
HISA-64	45.2	50.0	49.3	40.0	53.6	45.7	46.4
SCALE-128	45.2	51.0	56.3	46.4	53.6	45.7	49.6
SCALE-64	50.0	53.8	56.3	46.4	50.0	43.5	50.4

4.3 Matched-budget adaptive partitioning

On the 128K RULER replay, equal-summary comparisons at $N/8$, $N/32$, $N/64$, and $N/128$ all favor SCALE over fixed blocks (Figure 5): candidate recall increases by 2.37, 1.01, 0.49, and 0.32 points under the same 8,192-token budget, while score RelMSE decreases. The gain is largest at $N/8$ and remains positive at every displayed budget. Appendix F isolates direct splitting from split–merge.

4.4 End-to-end prefill and decode

On a 128K prompt with 128 generated tokens (Table 10), SCALE-128 reduces DSA TTFT from 62.90 to 38.92 seconds and E2E latency from 65.30 to 41.11 seconds. Matched-summary HISA-128 is close in TTFT (34.90 seconds), while SCALE-128 has lower TPOT. SCALE-64 raises pooled RULER from 65.3 to 82.1 and cuts DSA TTFT by 37.2%. Figure 6 reports DSA, HISA-64, and SCALE-64 as context length grows. At longer contexts, DSA TTFT grows much faster, whereas HISA-64 and SCALE-64 stay close to linear. Panel (c) shows cumulative prefill plus decode time on one 128K prompt while generation continues through 10K tokens.

4.5 Ablations

We ablate the partition criterion, split versus split–merge, minimum atom size, and λ-search precision (Table 6, Appendix F). At a matched $N/8$ split, P-key improves mean Top-2,048 recall by 1.77 points over uniform blocks and raises the fifth percentile from 31.93 to 35.66. At $N/64$, over-splitting to $N/8$ then Ward-merging beats both fixed-64 and a direct $N/64$ split.

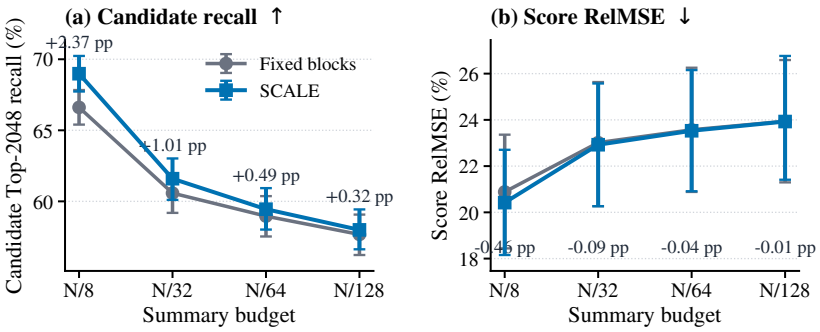


Figure 5: Matched-budget 128K RULER replay (624 paired queries). Both methods admit 8,192 candidates; labels show paired SCALE-minus-fixed changes and bars show 95% bootstrap CIs. SCALE starts from an $N/8$ split.

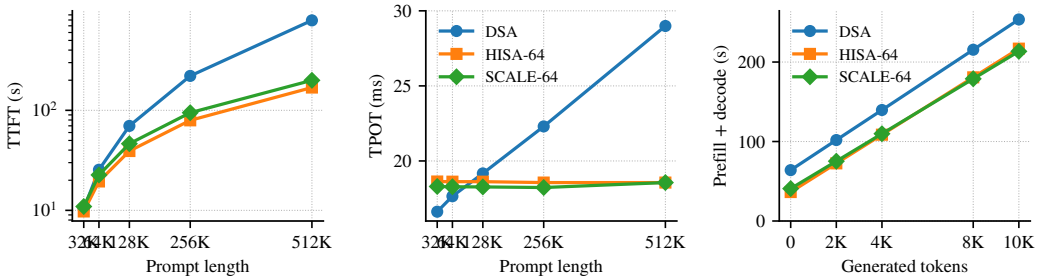


Figure 6: End-to-end context scaling on TP8 H20 with FP8 KV cache. (a) TTFT versus prompt length (log y). (b) Decode TPOT versus prompt length. (c) Cumulative prefill+decode time versus generated tokens on one 128K prompt. All panels compare DSA, HISA-64, and SCALE-64. Contexts above the model's native 160K limit in (a,b) are systems-only.

5 Related Work

Fixed-granularity sparse indexers. QSA (Qiu et al., 2026) compresses index keys into fixed micro-blocks; HISA (Xu et al., 2026) and LongCat-HI (Zan et al., 2026) score pooled blocks before token-level refinement. SCALE keeps this coarse-to-fine interface but allocates the summary budget non-uniformly according to key-space dispersion.

Adaptive key-space filtering. Quest bounds scores with fixed-page extrema, UNIQUE augments fixed-page means with dispersion, and AB-Sparse varies block sizes across heads (Tang et al., 2024; Deng et al., 2026; Liu et al., 2026b). SCALE uses key dispersion to place contiguous boundaries within one sequence and retains mean-key summaries.

Other sparse-attention systems. Static patterns, learned sparse attention, and KV-access methods change where attention looks or which keys stay resident, rather than adapting chunk granularity inside a resident index (Child et al., 2019; Beltagy et al., 2020; Jiang et al., 2024; Yuan et al., 2025; Lai et al., 2026; Fu et al., 2026; Ribar et al., 2024; Sun et al., 2025; Liu et al., 2026a; Qi et al., 2026; Zhao et al., 2026; Yang et al., 2026).

6 Conclusion

We introduce SCALE, a training-free adaptive-resolution hierarchical index that allocates a fixed summary budget according to query-independent key dispersion, using finer resolution for heterogeneous regions and coarser resolution for homogeneous ones. At matched summary and refinement budgets, SCALE improves candidate recall over fixed-block indexing by 0.32–2.37 percentage points. At the $N/64$ operating point, SCALE-64 raises pooled RULER accuracy from 65.3 to 82.1 over HISA-64, achieves the same LongBench-v2 point estimate as DSA (50.4), and reduces DSA end-to-end latency from 65.30 to 41.69 seconds.

AI USE STATEMENT

We used generative AI tools to assist with drafting and language editing of the manuscript, including grammar, wording, paragraph flow, and the clarity of technical descriptions, and with LaTeX formatting and figure/table layout. All AI-assisted outputs were reviewed and verified by the authors. The authors take full responsibility for the methodology, implementation, mathematical claims, experimental results, and final content of the manuscript.

REPRODUCIBILITY STATEMENT

We provide algorithmic and serving specifications in Appendices A–C, including pseudocode for partition construction, prefill, and decode. Appendix E reports the complete serving configuration, and Appendix H describes the measurement protocol and provenance used for quality and end-to-end evaluation. The proof of Proposition 1 is in Appendix B.

REFERENCES

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

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. In *International Conference on Learning Representations*, volume 2025, pp. 44169–44190, 2025.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Keqi Deng, Shaoshi Ling, Ruchao Fan, and Jinyu Li. UNIQUE: Universal top-k sparse attention for training-free inference and sparsity-aware training. *arXiv preprint arXiv:2605.27740*, 2026.

Yaosheng Fu, Guangxuan Xiao, Xin Dong, Song Han, and Oreste Villa. SparDA: Sparse decoupled attention for efficient long-context LLM inference. *arXiv preprint arXiv:2606.04511*, 2026.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 52481–52515, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5dfbe6f5671e82c76841ba687a8a9ecb-Abstract-Conference.html.

Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, Xiaolong Li, Haohai Sun, Haichao Zhu, Vito Zhang, et al. MiniMax sparse attention. *arXiv preprint arXiv:2606.13392*, 2026.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

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. *Advances in Neural Information Processing Systems*, 38: 54358–54385, 2026a.

Di Liu, Ruitian Wang, Chen Chen, Mingliang Gong, Yongjie Yuan, Han Zhao, Yu Feng, Quan Chen, and Minyi Guo. AB-Sparse: Sparse attention with adaptive block size for accurate and efficient long-context inference. *arXiv preprint arXiv:2605.12110*, 2026b.

Yanlin Qi, Xinhang Chen, Huiqiang Jiang, Qitong Wang, Botao Peng, and Themis Palpanas. ParisKV: Fast and drift-robust KV-cache retrieval for long-context LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=wxD4wTYQXt.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, et al. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: bandwidth-efficient llm inference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=KzACYw0MTV.

Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Yufei Xu, Fanxu Meng, Fan Jiang, Yuxuan Wang, Ruijie Zhou, Zhaohui Wang, Jiexi Wu, Zhixin Pan, Xiaojuan Tang, Wenjie Pei, Tongxuan Liu, di yin, Xing Sun, and Muhan Zhang. HISA: Efficient hierarchical indexing for fine-grained sparse attention. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=GnWqiwzNzK.

Xu Yang, Jiapeng Zhang, Dongyang Zhao, Guo Chen, and Zhuo Tang. Self-indexing kvcache: Predicting sparse attention from compressed keys. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 27675–27683, 2026.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, 2025.

Wen Zan, Jiaqi Zhang, Jianchao Tan, Hong Liu, Cunguang Wang, Xiang Li, Duyue Ma, Guanyu Wu, Yifan Lu, Fengcun Li, et al. Longcat sparse attention: Taming the lightning via streaming-aware hierarchical cross-layer indexing. *arXiv preprint arXiv:2608.01662*, 2026.

Zihan Zhao, Baotong Lu, Shengjie Lin, Yizou Chen, Jing Liu, Yanqi Zhang, Ziming Miao, Ming-Chang Yang, Haiying Shen, Qi Chen, et al. Unifying sparse attention with hierarchical memory for scalable long-context llm serving. *arXiv preprint arXiv:2604.26837*, 2026.

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.

Algorithm 1 SCALE exact-count partition construction.

Require: FP8 keys/scales (K, s); indexed length N; split target M_0; merge target T
1: Build or extend the dyadic Key-SSE forest
2: Fixed-budget Key-SSE split: search λ with parallel dynamic-program evaluations
3: Materialize the feasible leaf mask and split gains
4: Exact-count repair: radix-select gains and reach M_0 leaves
5: Emit intervals and their key sums and means
6: **while** leaf count $m > T$ **do**
7: Target-count Ward merge: compute costs and select eligible cheap edges
8: Form a deterministic non-overlapping matching
9: Merge without overshooting T; compact intervals, sums, and means
10: **end while**
11: Requantize interval means to FP8 summaries

A Algorithmic and serving details

Toy split–merge example. Figure 3 uses the scalar keys $z = [0, 0.5]^{\times 6} \| [3, 3.5]^{\times 2}$. The displayed Key-SSE values are $E_K([1, 16]) = 28$, $E_K([1, 8]) = 0.50$, $E_K([9, 16]) = 18.50$, and $E_K([9, 12]) = E_K([13, 16]) = 0.25$. At $\lambda = 0.125$, the two children of $[9, 16]$ each cost $F_c = 0.25 + 0.125 = 0.375$, so splitting costs 0.750, less than keeping the parent at 18.625. Its direct split gain is $18.50 - 0.25 - 0.25 = 18 > \lambda$. For $[1, 8]$, the possible children $[1, 4]$ and $[5, 8]$ also each have $E_K = 0.25$ and $F_c = 0.375$, but keeping costs only $0.50 + 0.125 = 0.625 < 0.750$. Its direct gain is zero, so it stays intact. At the root, splitting costs $0.625 + 0.750 = 1.375 < 28.125$. The resulting lengths $(8, 4, 4)$ meet the toy budget $M_0 = 3$. For merge target $T = 2$, the adjacent Ward costs are 0 and 18; removing the first boundary produces $[1, 12]$ and $[13, 16]$. Cell colors use $\tilde{z}_t = z_t/3.5$ and the same bins as Figure 1; splitting and merging never change a token's color. Key-SSE, λ, and Ward costs are computed from the unscaled z_t, so the numerical example retains a single consistent cost scale.

Searching the leaf penalty. The upper flow in Figure 3(a) maintains an infeasible lower endpoint λ_{lo} and a feasible upper endpoint λ_{hi}. With $\mathcal{R}$ the set of tree roots, initialize $\lambda_{\text{lo}} = 0$ and $\lambda_{\text{hi}} = 2 \max_{r \in \mathcal{R}} E_K(r) + 1$. The upper endpoint exceeds every possible split gain, leaving one interval per root; it is feasible for the configured budgets $M_0 \geq |\mathcal{R}|$. If the count at zero is already feasible, use $\lambda = 0$ directly.

Otherwise, each round evaluates the 16 interior candidates

$$\lambda_j = \lambda_{\text{lo}} + \frac{j}{17}(\lambda_{\text{hi}} - \lambda_{\text{lo}}), \qquad j = 1, \ldots, 16.$$

GPU programs evaluate Eq. (6) bottom-up in parallel over roots and candidate groups, reducing root counts to one $m(\lambda_j)$ per candidate. A device-side update retains the adjacent grid points around the first feasible count $m(\lambda_j) \leq M_0$, including the old endpoints when needed. Thus each active round shrinks the bracket by $17\times$. The exact path uses at most six rounds, with early stopping when the feasible endpoint's deficit is at most 0.05% of M_0; later captured launches return immediately without host synchronization. It selects the feasible endpoint and repairs any remaining deficit by repeatedly adding the available split with the largest Key-SSE reduction until exactly M_0 leaves are present. The GPU radix implementation reproduces this greedy order. "Exact" refers to the leaf count, not to a globally optimal solution of the count-constrained partition problem.

Relaxed fixed-shape schedule. The serving path uses two λ rounds and materializes $m_{\text{split}} \leq M_0$ leaves without exact repair. Two soft rounds merge non-overlapping pairs satisfying $\Delta_K(a, b) < 2\lambda$, followed by one target-count round. If that matching cannot fit the output capacity, at most $\lceil \log_2 \lceil M_0/T \rceil \rceil$ fixed pairing rounds merge only while $m > T$. Thus the schedule guarantees $m \leq T$ without host synchronization. Arrays have static capacity $64\lceil T/64 \rceil$; rows beyond the device-side logical count are normalized to $(N, 0)$ and ignored.

Algorithm 2 is the chunked-prefill loop of Figure 4(a.1–a.2). Algorithm 3 is one decode step with a fixed prompt partition.

Algorithm 2 Sparse prefill for query chunk $\mathcal{C}_i$ ($i \geq 1$).

Require: queries Q_i in $\mathcal{C}_i$; raw keys; previous partition P_{i-1}; budget C
if P_{i-1} is unavailable, too short, or chunk i is dense-final **then**
  Compute flat DSA logits and their Top-K IDs
else
  Wait for P_{i-1}'s ready event
  Score summaries and weighted-radix select C historical tokens
  Exactly score selected history and densely score the causal local window
  Apply split-row radix Top-K
end if
Launch the side-stream build of P_i and record its ready event
return request-relative Top-K token IDs

Algorithm 3 One SCALE decode step.

Require: query q; prompt partition P; generated length n; budget C
if a 64-token generated block has completed **then**
  Mean-pool, requantize, and append its interval summary
end if
Score summaries and reserve exact sink/tail tokens
Weighted-radix fill the remaining budget with clipped intervals
Exactly score the C raw candidates and return their Top-K IDs

B Proof of the Key-SSE score bound

Proof of Proposition 1. For every token t in chunk b, ReLU's 1-Lipschitz property, the triangle inequality, and Cauchy–Schwarz give

$$|I(i,t) - I(i,\mu_b)| \leq \sum_{h=1}^{H} |w_{i,h}| \left|q_{i,h}^{\top}(k_t - \mu_b)\right| \leq A_i \|k_t - \mu_b\|_2.$$

Squaring and averaging this inequality over $t \in b$ gives Eq. (4). Summing its squared form over all chunks in P and dividing by N gives Eq. (5). □

Requantized summaries. Proposition 1 uses the dequantized mean μ_b. The implementation stores a requantized mean $\widehat{\mu}_b$. Applying the same Lipschitz argument and the RMS triangle inequality gives

$$\left(\frac{1}{|b|}\sum_{t\in b} |I(i,t) - I(i,\widehat{\mu}_b)|^2\right)^{1/2} \leq A_i \left(\sqrt{\frac{E_K(b)}{|b|}} + \|\mu_b - \widehat{\mu}_b\|_2\right).$$

The second term is the error introduced by quantizing the summary and is separate from the Key-SSE partition objective.

C Partition invariants and implementation details

Coverage. The indexed prefix length is $N_{\text{indexed}} = \lfloor N/256 \rfloor \cdot 256$. Emitted intervals are sorted, non-overlapping, atom-aligned, and cover $[0, N_{\text{indexed}})$ exactly. Padding rows have zero length and never participate in selection. The suffix $[N_{\text{indexed}}, N)$ remains in the exact local region.

Rules for equal values. When the keep and split costs are equal, the dynamic program keeps the parent unsplit. Repair orders equal gains lexicographically by token start, tree level, and node index. Weighted selection orders equal summary scores by interval index. The optimized radix paths are tested against these reference orders. Final Top-K comparison is set-based because equal FP8 logits may be resolved by GPU atomics in different orders.

Target merge. Suppose a round begins with m leaves and target T. The implementation admits the $2(m - T)$ cheapest adjacent edges before non-overlap filtering. It then selects at most $m - T$ matched pairs, so a round cannot overshoot the target. Rows are compacted together with interval

Table 3: Partition-construction operator latency on one H20 (ms, median of 10 post-warm-up CUDA-event samples). All optimized split/merge boundaries match the PyTorch reference; incremental tree reuse is bitwise equal to a full raw-FP8 build.

Stage	Implementation	8K	16K	32K	64K	128K
Key-SSE tree	PyTorch materialized	0.475	0.478	0.607	1.060	1.941
Key-SSE tree	Raw-FP8 Triton	0.126	0.193	0.330	0.600	1.144
Key-SSE tree	Incremental Triton (+8K)	0.125	0.147	0.148	0.147	0.147
Split	PyTorch dynamic program+sort	8.178	8.233	8.255	8.285	8.238
Split	Fused dynamic program+radix	0.976	1.050	1.034	1.034	1.128
Target merge	PyTorch sort+compact	8.312	8.330	8.327	9.230	9.169
Target merge	Triton+radix	2.516	2.518	2.541	2.579	2.581

Table 4: Hierarchical-selection operators at the 128K production shape on one H20 (median CUDA-event time).

Operator	Unit	Baseline	Optimized	Speedup
Prefill leaf selection	ms	1.896	0.462	4.10×
Fine candidate scoring	ms	4.882	4.352	1.12×
Top-K over leaf+local	ms	1.993	1.195	1.67×
Decode weighted selection	μs	30.939	13.596	2.28×

sums and means; no key rescan is required for summaries. If a fixed round cap cannot reach T, that configuration is reported as incomplete. The relaxed variant instead follows its soft and target-count rounds with the fixed-capacity fallback in Section 3.3; its valid-row count may be below T, but it always fits the exported capacity.

FP8 summaries. Raw index keys use E4M3 values with one FP32 scale per token. Tree accumulation and Ward statistics use FP64 in the reference-compatible path. After merging, the interval total is divided by its length and requantized to one E4M3 vector plus one scale. Summary pages reuse the index-key page layout with 64 rows per page.

D Detailed operator tables

Section 4.1 validates each optimized operator against a functional reference. The two tables below report the corresponding kernel times on one H20, using CUDA-event medians after warm-up.

Table 3 measures partition construction. Incremental Key-SSE tree extension for an added 8K chunk stays at 0.147 ms from 16K to 128K, whereas a full raw-FP8 rebuild grows to 1.144 ms at 128K and a PyTorch materialized rebuild to 1.941 ms. The fused dynamic-program split is 7.3× faster than the PyTorch sort path at 128K (1.128 ms versus 8.238 ms), and the Triton+radix Ward merge is 3.55× faster (2.581 ms versus 9.169 ms). These construction kernels remain milliseconds, well below the seconds of TTFT in Table 10.

Table 4 reports hierarchical selection at the 128K production shape. Prefill leaf selection is 4.10× faster than the baseline (0.462 ms versus 1.896 ms), and decode weighted selection is 2.28× faster (13.6 μs versus 30.9 μs). Fine candidate scoring changes less (1.12×) because it is already dominated by exact token-level rescoring of the C=8,192 admitted tokens.

E Serving configuration

The primary configuration uses atom 1, 256-token roots, an $N/8$ split budget, an $N/64$ final summary budget, at most eight captured merge rounds, up to six λ rounds with 16 interior candidates each, candidate budget 8,192, sink 64, local tail 256, and decode indexing chunk 64. All 61 DeepSeek-V3.2 indexer layers use the same configuration. Partition construction uses a side CUDA stream and CUDA graph replay where the shape has already been captured.

Table 5: Claim-to-configuration map. Quality and latency are paired only within the same named configuration.

Evidence	Partition	Prefill policy	Interpretation
RULER / LongBench-v2 + E2E timing	SCALE-128	dense final	quality and latency
RULER / LongBench-v2	SCALE-64	dense final	higher-fidelity quality point
Matched candidate recall	S8/M64	replay only	partition fidelity

Table 6: Main ablations. Left: partition criteria at the $N/8$ split budget (Top-2,048 recall, mean/P5). Right: split–merge construction at the $N/64$ summary budget on 73 prompts; lower RelMSE is better.

Criterion ($N/8$ split)	Mean	P5
Uniform blocks	74.66	31.93
Per-dimension box	74.72	32.11
Key radius	78.88	29.92
P-key (ours)	76.43	35.66
P-qfull (query-cond.)	84.61	49.03

Construction ($N/64$)	Recall	RelMSE
Fixed-64 blocks	63.26	23.717
Direct $N/64$ split	63.45	23.709
SCALE-64 (split→merge)	**63.80**	**23.628**

The SCALE-128 arm uses an $N/32$ split budget and an $N/128$ final summary budget; SCALE-64 uses $N/8$ and $N/64$, respectively. Both use two 16-candidate λ steps, no exact repair, two soft merge rounds at $\alpha = 2$, one target-count Ward round, and two deterministic capacity-fallback rounds. Both named arms keep the final prompt chunk dense.

Chunked prefill timeline. Figure 4(a.1–a.2) uses one-based chunk indices: $\mathcal{C}_i = [(i-1)B, iB)$ for a complete $B = 8{,}192$-token server chunk. At each layer, $\mathcal{C}_1$ uses flat DSA because P_0 is absent. After its Top-K, a side stream builds P_1 over $[0, B)$ while the main stream continues MLA and later layers. Chunk $\mathcal{C}_2$ waits for and consumes P_1; its history equals the 8,192-token candidate budget, so this step performs no prefix pruning. The current causal window $[B, 2B)$ is still scored densely by the indexer. After Top-K, construction extends the root forest and repartitions $[0, 2B)$ into P_2 for $\mathcal{C}_3$. Historical tree statistics are reused; the next partition covers the entire completed-root prefix, including the just-processed chunk. The dense-final arm uses flat DSA scoring for the last prompt chunk and still constructs the final partition for decode. Decode binds that completed prompt partition once.

Tree reuse. The dyadic forest never crosses a 256-token root. When a prompt chunk adds new complete roots, all existing node energies are copied into their new level offsets and kernels are launched only for new roots. The subsequent λ search, repair, and merge still process the full forest because their budgets and boundaries can change globally.

Claim-to-configuration map. Table 5 records which measurements are paired with which named configuration.

The SCALE-128 headline joins its own downstream scores with its own serving measurements and is interpreted directly against DSA and HISA.

F Additional ablations

Table 6 collects the criterion and construction results cited in Section 4.5. The left pane compares partition energies at a matched $N/8$ split; the right pane compares fixed blocks, a direct $N/64$ split, and split–merge at the $N/64$ summary budget. The subsections below expand each comparison.

F.1 Direct final-budget construction

To isolate over-splitting before merging, we compare fixed-64, a direct $N/64$ Key-SSE split, and the $N/8 \rightarrow N/64$ split–merge path at identical summary and candidate budgets. The replay contains 73 independent RULER/LongBench-v2 prompts and bootstraps prompt-level means.

Table 7 reports the three constructions. Direct splitting improves mean recall by 0.191 points over fixed blocks. Resampling the 73 prompts gives a 95% bootstrap confidence interval of $[0.128, 0.259]$, which excludes zero. Split–merge adds another 0.353 points (interval $[0.285, 0.421]$) and lowers

Table 7: Direct-$N/64$ ablation. Lower RelMSE is better.

Partition	Construction	Recall (%)	RelMSE (%)
Fixed-64	uniform blocks	63.26	23.717
Direct $N/64$	Key-SSE split only	63.45	23.709
SCALE-64	split then Ward merge	**63.80**	**23.628**

Table 8: Offline criterion ablation at matched split count and candidate budget. P-qfull is not the serving method.

Criterion	Mean (%)	P5 (%)
Uniform K=8	74.66	31.93
Per-dimension box	74.72	32.11
Key variance	76.43	35.66
Key radius	78.88	29.92
P-qfull (ours, split only)	84.61	49.03

RelMSE by 0.081 points. The latter gain remains positive on RULER ($+0.268$), its 128K slice ($+0.229$), and LongBench-v2 ($+0.400$).

F.2 Offline criterion ablation

Before selecting P-key for serving, we replayed 748 teacher-forced query rows at 11 layers and compared several partition energies at a matched $N/8$ split budget. P-qfull scores each candidate split using the actual indexer scores of saved calibration queries, so it shows an optimistic ceiling for a query-aware partition. That computation needs a dense score against every token and cannot be performed before queries arrive; SCALE therefore does not use it. Table 8 restates this comparison: key-variance (P-key) raises mean recall from 74.66 to 76.43 and the fifth percentile from 31.93 to 35.66, while key radius has a higher mean but a worse tail (29.92). Table 9 coarsens the minimum atom from one token to four. P-key drops only from 76.43 to 75.93, whereas P-qfull drops from 84.61 to 77.67; the serving path therefore keeps a one-token atom.

G Offline exact-path λ precision

G.1 How many λ-bracketing rounds are needed?

This appendix records an offline precision study for the *exact* S8/M64 construction. Its solid curves use query-conditioned score energies, and the post-merge curve uses the earlier $\alpha\lambda$ threshold-merge prototype. They therefore do not evaluate the deployed P-key target-count merge or the relaxed S32/M128 schedule; only the dotted synthetic curves use key-space energies. We retain the study because it motivates the six-round cap used by the exact path.

The split has two parts that could each be blamed for its cost: bracketing the per-leaf penalty λ, and spending the leftover budget. Section 3 maintains a lower endpoint that yields more than M leaves and an upper endpoint that yields at most M leaves. Each round evaluates 16 interior λ values and retains the sub-bracket containing the leftmost feasible transition; it then repairs the remaining deficit by splitting the leaf with the largest Ward gain until the count is exactly M. The serving implementation runs both parts on the GPU: one fused dynamic program evaluates each candidate λ, a round shrinks the bracket by $17\times$, and the exact repair is one radix selection over the frontier without a host round trip. Ten steps take the bracket to a relative width of 10^{-12}. Since the leaf count is a step function of λ, the program at the chosen feasible λ returns $N_{\text{split}} \le M$ leaves and the repair creates the other $M - N_{\text{split}}$ (Figure 7a). The repair is exact with respect to the budget but greedy with respect to the objective, so it is not obvious how much of the search can be handed to it. We truncate the search at $S \in \{1, \dots, 10\}$ steps and compare against S=10.

Data. The sweep dump of 18 September 2026 stores, for 20 layers of DeepSeek-V3.2 (Liu et al., 2025) and 5 RULER (Hsieh et al., 2024) requests of about 32K tokens, the FP8 indexer keys and 16 query rows per request. We rebuild the indexer score of each saved row against every key and use the 16 rows as the calibration set, so the energies are query-conditioned indexer scores of the same form as in Section 4.1, but the rows are the saved decode rows rather than 32 strided prefill states.

Table 9: Offline effect of increasing the minimum atom from one to four tokens.

Criterion	Atom 1	Atom 4
Per-dimension box	74.72	74.71
Key variance	76.43	75.93
Key radius	78.88	76.71
P-qfull	84.61	77.67

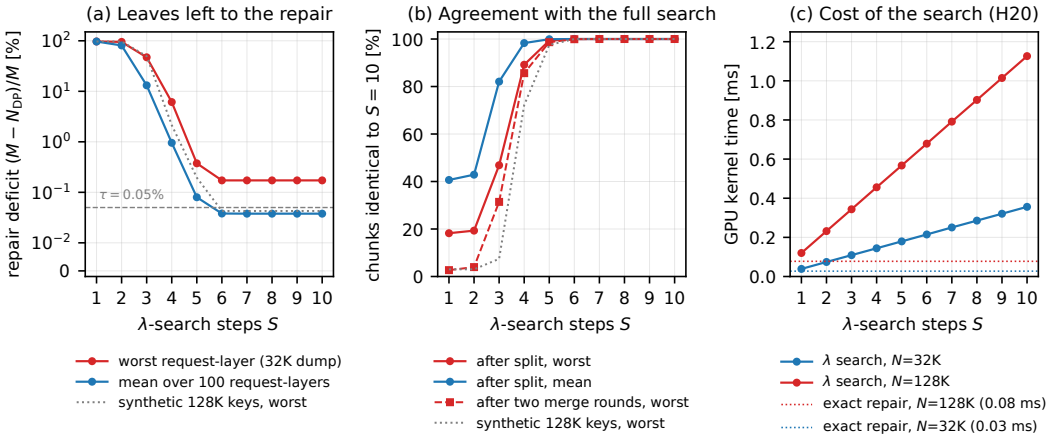


Figure 7: Truncating the λ search. S is the number of bracketing rounds; each round evaluates 16 interior λ values in parallel, with one dynamic program per value, and the reference $S{=}10$ brackets λ to a relative width of 10^{-12}. (a) Fraction of the $M{=}N/8$ chunks that the repair has to create, over 100 request-layers of the DeepSeek-V3.2 dump. (b) Fraction of chunks identical to the $S{=}10$ partition, after the split and after two synchronous rounds of the legacy threshold merge. The dotted gray curves repeat the worst case on synthetic 128K keys with key-space energies. (c) GPU kernel time of the search on an H20 against the single-pass exact repair; the $S{=}10$ point at 128K is the linear extrapolation because the measured value was inflated by another process sharing the GPU.

This gives 100 request-layer score fields with M between 4,000 and 4,064. Because the search only sees the tree energies, we also repeat the sweep on synthetic 128K FP8 keys with key-space energies, three seeds each of a segmented and a uniform key distribution, to check that the conclusion is not specific to 32K or to score energies.

Six rounds reproduce the full search. From $S{=}6$ on, λ moves by less than 7×10^{-5} relative to its $S{=}10$ value, the program returns the same leaf count, and the partition is identical on all 100 request-layers after the split and on all but one after two merge rounds, where two chunks differ because the merge threshold $\alpha\lambda$ moved with λ (Figure 7b). Rounds 7–10 refine a λ value that no longer changes any leaf. The residual deficit at $S{=}10$, at most 0.17% of M and 0.04% on average, is a property of the step function and not of the search; the repair spends it. The synthetic 128K sweep gives the same picture: identical partitions from six steps on, with a residual deficit of up to 0.32% on uniform keys.

Below six steps the repair is no longer a substitute. At $S{=}5$ the deficit is still small, at most 0.38%, but the worst request-layer already has 0.7% of its chunks changed after the split and 1.3% after merging. At $S{=}4$ the deficit reaches 6% and the worst-case agreement falls to 89% after the split and 86% after merging; on the synthetic 128K keys it falls to 72%. The reason is the one that motivated the dynamic program in Section 3. The repair ranks frontier leaves by their immediate Ward gain, whereas the Lagrange program also credits the gains that become reachable below a split. When the deficit is a handful of leaves the two rankings agree; when the repair has to make hundreds of decisions it drifts away from the Lagrangian optimum. Cheap repair therefore does not license an arbitrarily coarse λ bracket.

Stopping on the leaf count. The natural stopping rule is not the width of the bracket but the deficit it leaves: stop once the feasible end of the bracket returns $N_{\text{split}} \geq (1-\tau)M$ leaves. The GPU implementation records that count in the same kernel that updates the bracket and raises a device-side flag, so later steps return immediately without changing the captured launch sequence. With $\tau{=}0$,

Table 10: End-to-end timing for a 128K prompt and 128 generated tokens (TP8, batch one, CUDA graphs). All hierarchical methods use an 8,192-token refinement budget.

	Prefill / end-to-end			Decode		
Method	TTFT (s)	Δ vs. DSA	E2E (s)	TPOT (ms)	Output tok/s	Δ vs. DSA
DSA	62.90	+0.0%	65.30	18.85	53.1	+0.0%
HISA-64	35.61	-43.4%	37.84	17.56	56.9	-6.8%
HISA-128	34.90	-44.5%	37.12	17.42	57.4	-7.6%
SCALE-64	39.48	-37.2%	41.69	17.46	57.3	-7.3%
SCALE-128	38.92	-38.1%	41.11	17.20	58.1	-8.7%

which accepts only an exact hit $N_{\text{split}}{=}M$, the flag fires on 23 of the 100 request-layers by step 5 and on 35 by step 6. With $\tau{=}0.05\%$ and a cap of six steps the search averages 5.4 steps and the partition is identical to $S{=}10$ on every request-layer; $\tau{=}0.1\%$ saves another 0.2 steps on average and changes up to 0.2% of the chunks on a few request-layers. The exact path uses six steps with $\tau{=}0.05\%$. The relaxed S32/M128 path deliberately uses two fixed rounds and no repair; its quality and timing are reported as a separate systems configuration, not inferred from this precision study.

What is saved. Figure 7c puts the cost in absolute terms. One bracketing round is 0.035 ms of kernel time at 32K and 0.112 ms at 128K on an H20, so going from ten to six rounds saves 0.14 and 0.45 ms; the exact repair costs 0.03 and 0.08 ms regardless of the deficit, less than one bracketing round. These are kernel times of the split stage inside the serving prefill path, not time to first token, and the split stage is not the largest part of that path. The ablation settles the precision question rather than the systems question: the λ bracket needs about six 17-way rounds, after which the exact repair, not further search, is the right way to spend the leftover budget.

H Measurement protocol and provenance

Operator timing. Each optimized operator is validated against a semantic reference before timing. Partition-construction stages use 10 measured CUDA-event samples after warm-up; selection operators report the median CUDA-event time after warm-up. CUDA events surround only the operation under test, followed by an explicit synchronization. We report the median over those samples.

End-to-end timing. Each arm runs in a fresh server process with no competing GPU process. The model, image, tensor parallelism, context limit, static memory fraction, chunk size, request order, and input IDs are fixed. TTFT is client submission to the first streamed token. TPOT is the elapsed time between first and final output tokens divided by the number of output-token intervals. CUDA-graph and eager measurements are never mixed in the same decode ranking.

Quality. RULER and LongBench-v2 predictions are stored per prompt with source ID, prompt length, generated text, score, and elapsed time. Comparisons are paired on common source IDs. Every generated table is accompanied by a JSON provenance record containing the source tree hash, launch arguments, environment delta, raw result paths, hardware, and generation timestamp.

I End-to-end serving measurements

Table 10 is the 128K serving point used in Section 4.4: one 128K prompt and 128 generated tokens on TP8 H20 with FP8 KV cache and CUDA graphs. SCALE-128 reduces DSA TTFT from 62.90 s to 38.92 s (a 38.1% reduction) and end-to-end latency from 65.30 s to 41.11 s, with decode TPOT 17.20 ms versus 18.85 ms. Matched-summary HISA-128 is close in TTFT (34.90 s); SCALE-128 has the lower TPOT of the two hierarchical methods. SCALE-64 raises pooled RULER accuracy from 65.3 to 82.1 over HISA-64 while cutting DSA TTFT by 37.2% (62.90 s to 39.48 s).