

HISA: Efficient Hierarchical Indexing for Fine-Grained Sparse Attention

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, Muhan Zhang[†]
<https://github.com/MuLabPKU/TransArch>

Abstract

Token-level sparse attention mechanisms, exemplified by DeepSeek Sparse Attention (DSA), achieve fine-grained key selection by scoring every historical key for each query through a lightweight indexer, then computing attention only on the selected subset. While the downstream sparse attention itself scales favorably, the indexer must still scan the entire prefix for every query, introducing an $\mathcal{O}(L^2)$ per-layer bottleneck that grows prohibitively with context length. We propose **HISA** (**H**ierarchical **I**ndexed **S**parse **A**ttention), a plug-and-play replacement for the indexer that rewrites the search path from a flat token scan into a two-stage hierarchical procedure: (1) a block-level coarse filtering stage that scores pooled block representations to discard irrelevant regions, followed by (2) a token-level refinement stage that applies the original indexer exclusively within the retained candidate blocks. HISA preserves the token-level top- k index interface consumed by the downstream Sparse MLA operator and requires **no additional training**. In end-to-end serving with DeepSeek-V3.2, HISA improves throughput by $1.65\times$ and time to first token by $1.70\times$ over the official DSA implementation, despite the HISA-specific operators being implemented in TileLang/Triton while the official DSA baseline uses highly optimized hand-tuned CUDA kernels. With matched TileLang implementations of both indexers, controlled microbenchmarks show up to $3.75\times$ **indexer speedup** at 64K context. On Needle-in-a-Haystack and LongBench, we directly replace the indexer in DeepSeek-V3.2 and GLM-5 with our HISA indexer, without any finetuning. HISA closely matches the original DSA in quality, while substantially outperforming block-sparse baselines.

1 Introduction

Serving large language models (LLMs) (OpenAI, 2026; Anthropic, 2026; Google DeepMind, 2025; Meta, 2025; Qwen, 2026; DeepSeek-AI, 2024; MiniMax et al., 2025; Moonshot AI, 2025) over long contexts remains a central systems challenge. As context windows grow from 128K to 1M tokens and beyond—driven by demands for agentic multi-turn reasoning, long-document understanding, and native multimodal processing—the quadratic cost of self-attention becomes a dominant bottleneck in both prefill latency and memory consumption (Dao et al., 2022; Dao, 2023).

A productive line of work tackles this challenge through *sparse attention*: instead of attending to all key-value pairs, each query selects a small subset of the most relevant tokens and computes attention only over that subset. DeepSeek-V3.2 (DeepSeek-AI, 2025) adopts a *token-level* sparse attention paradigm, in which a lightweight *indexer* scores every historical token for each query, selects the top- k highest-scoring keys, and forwards only those keys to a downstream Sparse Multi-Head Latent Attention (Sparse MLA). This design has also been adopted in GLM-5 (GLM-5-Team, 2026) and provides strictly finer-grained selection

*Equal contribution.

[†]Corresponding author: muhan@pku.edu.cn

than block-level methods such as MoBA (Lu et al., 2025) and Native Sparse Attention (Yuan et al., 2025).

However, the token-level sparse paradigm introduces a subtler bottleneck. Although the downstream attention is sparse and cheap, the indexer itself must score every token in the prefix for every query. Concretely, if the prefix length is L and the indexer runs once per query per layer, the per-layer indexing cost is $\mathcal{O}(L^2)$ —the same asymptotic scaling as dense attention. As context lengths push toward 128K or 1M tokens, the indexer can transition from a negligible overhead into the dominant cost component.

This observation motivates a natural question: *can we reduce the indexer’s search cost without changing the final sparse attention pattern it produces?* In other words, can we rewrite the search path while preserving the search result?

We answer affirmatively with **HISA** (Hierarchical Indexed Sparse Attention). HISA replaces the flat, full-prefix token scan with a two-stage hierarchical search (shown in Figure 1):

1. **Block-level coarse filtering.** The prefix is partitioned into contiguous blocks of size B . A pooled representative vector is computed for each block via mean pooling over its constituent indexing keys. The query scores all $\lceil L/B \rceil$ block representatives and retains only the top- m blocks, immediately pruning the majority of the prefix from further consideration.
2. **Token-level refinement.** The token-level indexer then scores at most mB tokens from the candidate blocks using the same scoring mechanism as the original DSA indexer, except that the candidate pool is restricted to the tokens within the selected blocks rather than the full set of L tokens considered in DSA. The final top- k token set is then selected from this reduced candidate pool.

Crucially, HISA produces outputs with the same structure as the original DSA indexer: for each query, a set of k token indices. As a result, the downstream Sparse MLA operator remains entirely unchanged. HISA is therefore a **drop-in replacement** that requires no retraining or architectural changes to the attention mechanism, and leaves the original token-level KV cache layout unchanged. HISA maintains pooled indexing-key summaries only for its coarse filter. The per-query indexing complexity drops from $\mathcal{O}(L)$ to $\mathcal{O}(L/B + mB)$, and the per-layer cost drops from $\mathcal{O}(L^2)$ to $\mathcal{O}(L^2/B + LmB)$.

Our contributions are as follows:

- We identify the indexer as an emerging bottleneck in token-level sparse attention systems and formalize the problem of **search-path optimization** for sparse indexers.
- We propose HISA, a hierarchical block-to-token indexing strategy that is training-free, operator-compatible, and asymptotically faster than the flat indexer.
- We provide optimized TileLang GPU kernel implementations for both stages of HISA and demonstrate $2\text{--}4\times$ kernel-level speedup at 64K contexts, as well as up to $1.65\times$ end-to-end throughput and $1.70\times$ TTFT speedup in an SGLang serving stack.
- We empirically validate that HISA achieves performance comparable to the original DSA on the Needle-in-a-Haystack and LongBench benchmarks.

2 Related Work

Block sparse attention. Block sparse attention partitions sequences into fixed-size blocks and restricts computation to selected blocks, mapping naturally to GPU tiled matrix multiplications. This design is hardware-friendly, but all tokens within a block must be retained or discarded together. Among training-free methods, MInference (Jiang et al., 2024) profiles each head offline and assigns one of several sparse patterns at inference time; FlexPrefill (Lai et al., 2025) estimates block scores online and selects blocks by a cumulative-attention threshold; XAttention (Xu et al., 2025) uses antidiagonal sums as an $\mathcal{O}(B)$ proxy for block importance; and SpargeAttention (Zhang et al., 2025) applies a two-stage online filter to

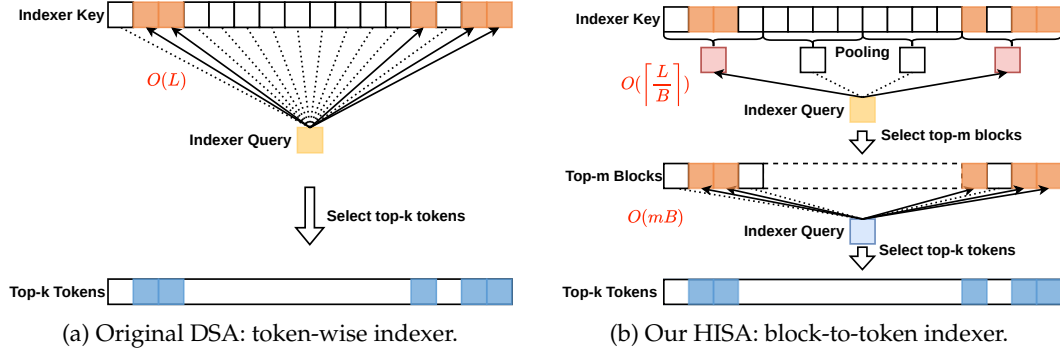


Figure 1: Comparison of the DSA token-wise indexer (left) and our HISA hierarchical block-level coarse filter followed by token-level refinement (right). Both produce the same data structure—a per-query set of k token indices—consumed by the downstream Sparse MLA operator.

skip low-importance regions during matrix multiplication and softmax. Among trainable methods, MoBA (Lu et al., 2025) uses mixture-of-experts-style routing over blocks, while NSA (Yuan et al., 2025) combines compression, selection, and sliding-window branches to cover different dependency scales. Their common limitation is block granularity: they cannot capture token-level importance differences within a selected block. HISA also introduces a block-level stage, but only as a fast pre-filter before token refinement; its final sparse pattern remains fine-grained and token-wise, as in DSA.

Token sparse attention. Token-level methods offer finer selection but face the challenge of efficient importance estimation. SnapKV (Li et al., 2024) uses an observation window at the end of the prompt to select important KV positions for subsequent decoding, but ignores layer- and query-specific variation. KV cache eviction methods—such as H₂O (Zhang et al., 2023), which combines cumulative attention with recency, and TOVA (Oren et al., 2024), which evicts the lowest-scoring cached token under the latest query—maintain a fixed-size cache but irreversibly lose evicted tokens. LazyLLM (Fu et al., 2024) progressively prunes tokens across layers during prefill, so early pruning mistakes cannot be corrected later in the same forward pass. DSA (DeepSeek-AI, 2025) instead scores every prefix token with a lightweight indexer and selects top- k tokens per query, achieving fine-grained sparsity at the cost of $O(L^2)$ per-layer indexing overhead. IndexCache (Bai et al., 2026) reduces this cost by reusing indices across nearby layers, although its benefit depends on cross-layer similarity in sparse patterns.

Hierarchical sparse attention. Hierarchical attention dates back to Yang et al. (2016), who introduced a two-tier word-and-sentence network for document classification. Among recent sparse methods, NSA (Yuan et al., 2025) and InfLLM-V2 (Zhao et al., 2026) can both be viewed as two-level designs: they score block-level summaries globally and activate finer-grained sparse attention only within selected blocks. Twilight (Lin et al., 2025) uses quantized keys for coarse scoring and then applies hierarchical top- p pruning, while Double-P (Ni et al., 2026) clusters the KV cache, scores cluster centroids, refines computation within selected clusters, and approximates low-score clusters with their centroids. HISA follows the same coarse-to-fine spirit but with a different goal: it combines a hardware-friendly block-level indexer with a fine-grained token-level indexer to accelerate DSA, achieving both high efficiency and strong selection quality on DeepSeek-V3.2 and GLM-5.

3 Preliminary

We briefly review DeepSeek Sparse Attention (DSA) as used in DeepSeek-V3.2 (DeepSeek-AI, 2025). DSA consists of two components: a **token-wise Indexer** and **Sparse MLA**.

Indexer in DSA. Let L denote the causal prefix length for a query position t . The indexer maintains lightweight indexing keys $\mathbf{k}_s^I$, indexing queries $\mathbf{q}_{t,j}^I$ for H^I indexing heads, and per-head gating weights $w_{t,j}^I$. The relevance score between query t and key s is defined as

$$I_{t,s} = \sum_{j=1}^{H^I} w_{t,j}^I \cdot \text{ReLU}(\mathbf{q}_{t,j}^I \cdot \mathbf{k}_s^I). \quad (1)$$

The indexer then selects the top- k token indices,

$$\mathcal{T}_t = \text{TopK}(I_{t,:}, k), \quad (2)$$

which are passed to the downstream Sparse MLA operator. Since the scoring cost for each query is $\mathcal{O}(L)$ over the full prefix, the total cost across all queries in a layer is $\mathcal{O}(L^2)$.

Sparse MLA in DSA. Following the DeepSeek-V3.2 design, Sparse MLA adopts the MQA mode of MLA, in which each token stores a single latent key-value entry shared across all query heads for efficiency. Let $\mathbf{c}_s$ denote the latent MLA entry associated with token s . Given the selected token set $\mathcal{T}_t$, Sparse MLA computes attention for query token t only over the selected latent entries, rather than over the full prefix:

$$\mathbf{u}_t = \text{Attn}(\mathbf{h}_t, \{\mathbf{c}_s \mid s \in \mathcal{T}_t\}). \quad (3)$$

As a result, the main attention cost is reduced from dense $\mathcal{O}(L^2)$ to sparse $\mathcal{O}(Lk)$. For our purposes, the key observation is that the interface between the two components is precisely the selected token set $\mathcal{T}_t$: HISA replaces only the indexer search path, while leaving the downstream Sparse MLA operator unchanged.

4 Method

4.1 HISA: Hierarchical Indexed Sparse Attention

As shown in Figure 1, HISA replaces the flat prefix scan with a two-stage coarse-to-fine search. The final output remains an identical per-query token set $\mathcal{T}_t^H$ of size k , consumed by the original Sparse MLA operator.

Block partitioning and pooled keys. The prefix of length L is partitioned into $M = \lceil L/B \rceil$ contiguous, causally valid blocks $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_M$, where B is the block size. For each block, a representative key is constructed via mean pooling over its indexing keys:

$$\tilde{\mathbf{k}}_b^I = \text{Pool}\left(\left\{\mathbf{k}_s^I \mid s \in \mathcal{B}_b\right\}\right). \quad (4)$$

These representative keys serve exclusively as coarse-grained proxies for block-level scoring and leave both the token-level indexing keys consumed by the second stage and the KV states consumed by Sparse MLA unchanged, thereby making HISA a plug-and-play replacement. In practice, these representative keys can be incrementally maintained alongside the KV cache with negligible overhead.

Stage 1: Block-level coarse filtering. For query position t , HISA reuses the same indexing query representations $\mathbf{q}_{t,j}^I$ and gating weights $w_{t,j}^I$ as DSA, but scores the **pooled representative keys** instead of individual token keys:

$$J_{t,b} = \sum_{j=1}^{H^I} w_{t,j}^I \cdot \text{ReLU}(\mathbf{q}_{t,j}^I \cdot \tilde{\mathbf{k}}_b^I). \quad (5)$$

The top- m blocks are selected:

$$\mathcal{C}_t = \text{TopK}(J_{t,:}, m), \quad (6)$$

and the candidate token set is the union of all tokens in the selected blocks:

$$\Omega_t = \bigcup_{b \in \mathcal{C}_t} \mathcal{B}_b. \quad (7)$$

All block selections strictly respect the causal mask: only blocks that precede the query position t , together with the block containing position t , are considered eligible. Following MoBA (Lu et al., 2025), the first and the last blocks are **always** included in $\mathcal{C}_t$, as they contain the attention sink and local contexts. This forced inclusion also simplifies boundary handling during batched prefill with packed sequences of varying lengths, where a single block may straddle the boundary between two sequences.

Stage 2: Token-level refinement. Within the selected candidate set Ω_t , the token-level indexer computes scores using the same scoring mechanism as in the original DSA (Eq. 1):

$$I_{t,s} = \sum_{j=1}^{H^I} w_{t,j}^I \cdot \text{ReLU}(\mathbf{q}_{t,j}^I \cdot \mathbf{k}_s^I), \quad s \in \Omega_t. \quad (8)$$

Then the top- k tokens are selected as final tokens:

$$\mathcal{T}_t = \text{TopK}(\{I_{t,s} \mid s \in \Omega_t\}, k). \quad (9)$$

To ensure that the candidate pool is sufficiently large to select k tokens, the feasibility constraint $mB \geq k$ must be satisfied. Given the selected token set $\mathcal{T}_t$, Sparse MLA is executed following the same computation as in the original DSA. Algorithm 1 provides the complete pseudocode for the HISA indexer.

Boundary behavior. Three regimes arise depending on the relationship between the effective prefix length t , the candidate capacity mB , and the budget k :

- When $t \leq k$, all prefix tokens are selected and HISA is equivalent to dense attention.
- When $k < t \leq mB$, the coarse filter selects all blocks (since $m \geq M$), and Stage 2 reduces the set to k tokens. HISA is equivalent to the original DSA indexer.
- When $t > mB$, the coarse filter performs non-trivial block pruning, activating HISA’s hierarchical advantage, which becomes increasingly pronounced as the sequence length grows.

The third regime is precisely the long-context setting where HISA provides its efficiency gains.

4.2 Complexity Analysis

Assuming that the pooled representative keys are maintained incrementally, the per-query indexing cost of HISA consists of scoring $\lceil L/B \rceil$ block representatives (Stage 1) and scoring at most mB candidate tokens (Stage 2):

$$\mathcal{O}\left(\frac{L}{B} + mB\right). \quad (10)$$

Summing over all L queries within a layer yields:

$$\mathcal{O}\left(\frac{L^2}{B} + LmB\right), \quad (11)$$

compared to $\mathcal{O}(L^2)$ for the original DSA indexer. The design introduces a clear trade-off: larger B reduces the cost of the coarse-filtering stage but makes each block a coarser proxy; smaller m improves efficiency but increases the risk of missing relevant blocks. When $m \ll M$ and $B \ll L$ —the regime of ultra-long contexts with a selective coarse filter—the reduction is substantial. Conversely, as m approaches M , HISA degrades gracefully toward the DSA baseline.

As modern LLMs increasingly adopt context windows of 128K or even 1M tokens to support advanced agent capabilities and native multimodal reasoning, HISA’s asymptotic advantage translates directly into practical speedups.

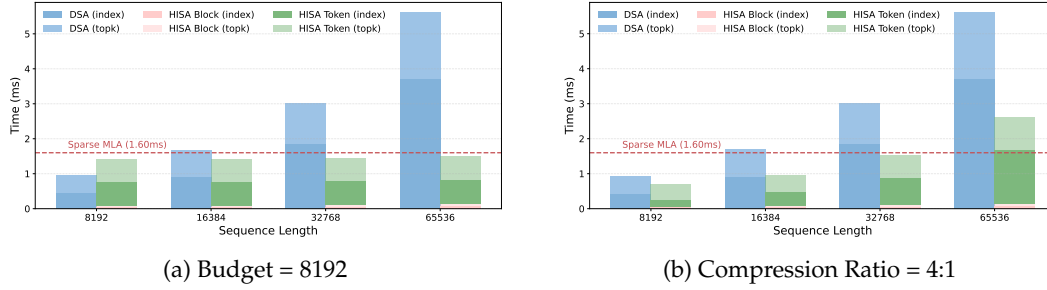


Figure 2: Kernel-level latency comparison of the indexer between the original DSA (flat token scan) and HISA (hierarchical block-to-token indexing). In the left panel, the block size is fixed to $B = 128$ and the maximum number of selected blocks is $\text{top-}m = 64$. In the right panel, $B = 128$ and the number of selected blocks is adjusted for each sequence length to maintain a fixed compression ratio of $M:m = 4:1$. The red dashed horizontal line marks the latency of the (TileLang) Sparse MLA operator, which stays roughly constant across context lengths, for reference. These measurements isolate indexer time; end-to-end results, including Sparse MLA, KV-cache management, scheduling, batching, and decoding, are reported in Section 5.2.

5 Experiments

We evaluate HISA along five main axes: (1) kernel-level latency, (2) end-to-end serving performance, (3) retrieval accuracy on Needle-in-a-Haystack, (4) downstream task performance on LongBench, and (5) token-selection consistency. Appendix C additionally reports hyperparameter sensitivity and a failure-case analysis. Throughout the evaluation, we compare three indexing strategies:

- **DSA (original)**: the full-prefix token-level indexer as described in Section 3.
- **Block-Sparse**: a block-level-only baseline that selects $\text{top-}m$ blocks and attends to all tokens within those blocks (i.e., Stage 1 only, without token-level refinement).
- **HISA**: the hierarchical block-to-token indexer proposed in this work.

Both HISA and Block-Sparse are *training-free*: they are applied at inference time by replacing the indexer module, with no fine-tuning or architectural modification.

5.1 Kernel-Level Speedup

Figure 2 compares the indexer kernel latency of the original DSA and HISA across context lengths from 8K to 64K tokens. Both implementations use TileLang (Wang et al., 2025) kernels, with DSA following the official reference implementation.¹ The HISA kernel is decomposed into two stages: block-level filtering and token-level refinement within the selected candidate blocks. The configuration is as follows: query lens = 1024, final $\text{top-}k = 2048$ tokens, block size $B = 128$, and two choices for the maximum number of selected blocks. All comparisons are conducted on an NVIDIA A100 GPU. These results are measured at the *indexer kernel* level and do not directly reflect end-to-end serving throughput, which also depends on the Sparse MLA operator, KV cache management, and other system components.

With 2048 selected tokens, the Sparse MLA operator consistently costs about 1.6 ms, while the indexer reaches 5.6 ms at 64K context length. This suggests that the main performance bottleneck in DSA lies in the indexer rather than in Sparse MLA itself. Accordingly, this section isolates indexer latency to characterize this bottleneck in a controlled setting; Section 5.2 reports the corresponding end-to-end serving results. At 64K context length, HISA delivers an approximately $2.16\times$ speedup with a 4:1 first-stage compression ratio (corresponding

¹<https://github.com/tile-ai/tilelang/tree/main/examples/deepseek.v32>

Table 1: End-to-end serving performance on DeepSeek-V3.2 in SGLang ($8 \times H200$, $TP=8$, 128K input and 512 output tokens per request). HISA achieves $1.65 \times$ throughput and $1.70 \times$ TTFT speedup despite its TileLang/Triton operators being less mature than the official DSA baseline’s hand-tuned CUDA kernels; the measured speedups therefore understate HISA’s potential at matched kernel maturity.

Indexer	Throughput (tok/s)	E2E latency (ms)	TTFT (ms)	TPOT (ms)
DSA (official)	5868.43	143764.47	106449.81	73.02
HISA ($B=128, m=64$)	9712.21	86718.06	62722.91	46.96
Speedup	$1.65 \times$	$1.66 \times$	$1.70 \times$	$1.55 \times$

to a 16K candidate budget), and up to $3.75 \times$ speedup under a fixed 8K budget. Although HISA adds a block-level filtering stage, this stage operates only on pooled block summaries of size $\lceil L/B \rceil$, which is far smaller than the full token sequence. Moreover, under a fixed 8K budget, the second-stage cost remains nearly constant because both the input and output lengths are fixed, making the computation graph easier to optimize and further improving inference speed.

5.2 End-to-End Serving Performance

We evaluate HISA in the full SGLang serving stack using DeepSeek-V3.2, against the official DeepSeek-V3.2 DSA implementation in SGLang as baseline; Appendix B.1 details the hardware and workload configuration. The baseline uses DeepSeek’s hand-tuned CUDA kernels, including `deep_gemm` for MQA indexer scoring, while HISA retains the optimized downstream `flashmla_sparse` path and replaces only the indexer; its newly introduced block pooling and sparse-MQA operators are currently implemented in TileLang or Triton and have not received the same degree of low-level tuning as the baseline.

We measure end-to-end serving on SGLang rather than vLLM, which we use for the quality evaluations in Sections 5.3 and 5.4, because SGLang’s serving architecture more readily accommodates HISA’s Stage-1 block-pooled key cache alongside the original token-level KV cache. Appendix B.2 gives the full reasoning, and Appendix C.2 reports a LongBench cross-check confirming that HISA’s quality conclusions transfer between the two stacks.

Table 1 shows that HISA improves all system-level metrics: throughput, end-to-end latency, time to first token (TTFT), and time per output token (TPOT). Unlike the kernel microbenchmark, this measurement includes Sparse MLA, KV-cache management, scheduling, batching, and autoregressive decoding. Since the HISA-specific operators have not yet received CUDA-level tuning comparable to the official DSA kernels, these end-to-end gains are a conservative estimate of HISA’s benefit at matched kernel maturity. The results nevertheless demonstrate that reducing index-selection cost translates into end-to-end serving gains on top of an optimized sparse-attention backend.

The gains persist across request concurrencies; detailed scaling results are provided in Appendix C.1.

5.3 Needle-in-a-Haystack

The Needle-in-a-Haystack (NIAH) test (Kamradt, 2023) evaluates a model’s ability to retrieve a specific fact (the “needle”) embedded at a controlled position within a long distractor context (the “haystack”). We evaluate DeepSeek-V3.2 with its original DSA indexer replaced by HISA (4:1 ratio) and block indexer, without any additional training, over context lengths ranging from 8K to 648K tokens and needle insertion depths ranging from 0% (beginning) to 100% (end).

Figure 3 presents the retrieval accuracy heatmaps. The original DSA achieves near-perfect retrieval across all context lengths and needle positions (Figure 3a). HISA closely matches this performance (Figure 3c), with only marginal degradation at extreme lengths and depths,

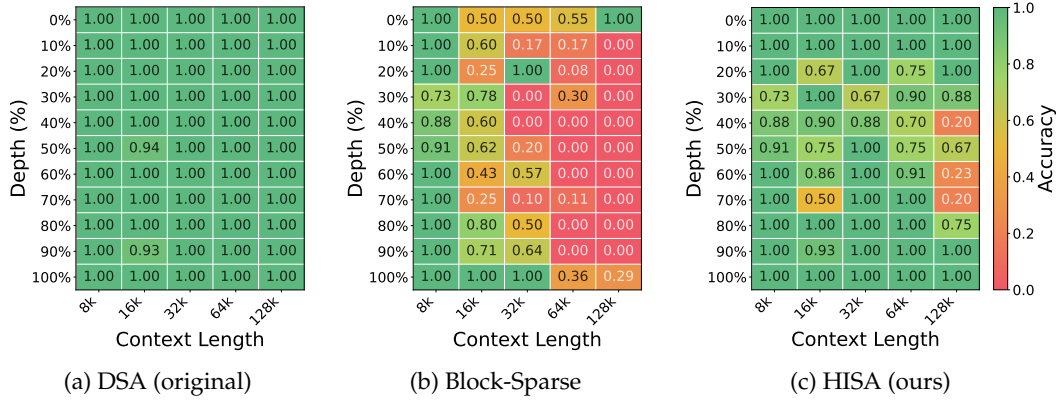


Figure 3: Needle-in-a-Haystack retrieval accuracy heatmaps for DeepSeek-V3.2 under three indexing strategies. The x -axis denotes the context length (8K–128K), and the y -axis denotes the needle depth (0%–100%). Shades closer to green indicate higher retrieval accuracy.

Table 2: LongBench results for DeepSeek-V3.2 and GLM-5 under different indexing strategies. All sparse methods are applied at inference time without additional training. Scores are averaged across sub-tasks within each category. Task abbreviations: **SQA** = Single-Document QA, **MQA** = Multi-Document QA, **Sum** = Summarization, **FS** = Few-shot Learning, **Syn** = Synthetic Retrieval, **Code** = Code Completion.

Model	Indexer	SQA	MQA	Sum	FS	Syn	Code	Avg.
DeepSeek-V3.2	DSA	50.89	52.66	22.11	62.24	<u>69.83</u>	48.56	51.05
	Block	48.36	49.76	21.90	59.45	<u>68.67</u>	49.09	49.54
	HISA	<u>49.17</u>	<u>51.96</u>	22.13	<u>61.62</u>	70.83	48.99	<u>50.78</u>
GLM-5	DSA	<u>41.23</u>	27.89	18.39	<u>63.20</u>	<u>68.84</u>	56.53	<u>46.01</u>
	Block	38.35	24.29	16.95	60.64	60.49	55.29	42.67
	HISA	42.45	<u>27.62</u>	<u>17.90</u>	63.78	69.35	56.79	46.32

suggesting that HISA rarely discards blocks containing the target information. In contrast, the Block-Sparse baseline (Figure 3b) exhibits noticeable accuracy degradation, particularly when the needle is located in the middle of the context where block-level selection is least reliable. This result underscores the value of hierarchical selection. Block-sparse methods often waste budget on unimportant tokens within selected blocks while overlooking truly critical tokens. HISA, in contrast, refines the selection at the token level after block retrieval, allowing it to preserve important tokens more accurately and achieve efficient token-wise sparsity.

5.4 LongBench Evaluation

LongBench (Bai et al., 2024) is a comprehensive benchmark for long-context understanding, covering single-document QA, multi-document QA, summarization, few-shot learning, and synthetic retrieval tasks. We evaluate DeepSeek-V3.2 (DeepSeek-AI, 2025) and GLM-5 (GLM-5-Team, 2026) under three configurations: the original DSA indexer, HISA, and Block-Sparse Attention. For a fair comparison, all three configurations ultimately retain 2048 tokens for computation. Specifically, Block-Sparse Attention directly selects 16 blocks of size 128 (i.e., $128 \times 16 = 2048$ tokens). HISA first selects 64 blocks of size 128 (i.e., $128 \times 64 = 8192$ tokens), and then further refines them through token-level selection to 2048 tokens.

Table 2 summarizes the results. Across both models and all task categories, HISA achieves performance very close to that of the original DSA. Notably, HISA consistently surpasses DSA on the Synthetic tasks, and on GLM-5 it even attains a higher average score. By

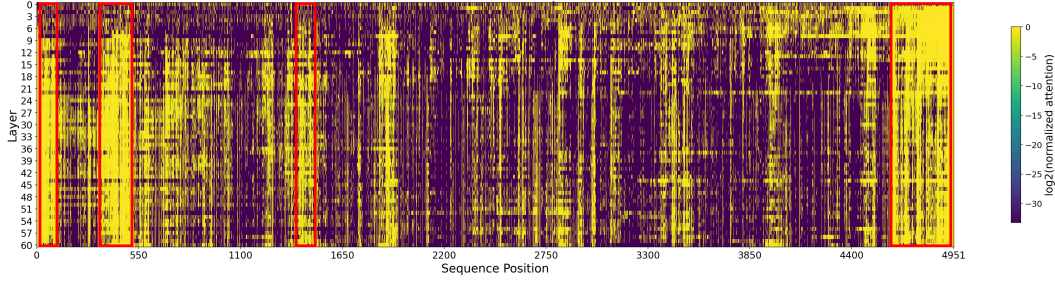


Figure 4: Visualization of Attention Distribution.

contrast, the Block-Sparse baseline, which does not include token-level refinement, exhibits a substantially larger performance gap. This is particularly apparent on the Synthetic tasks for GLM-5, where its score declines by 8.35 points.

5.5 Visualization of Attention Scores

To analyze the structural properties of attention in long-context generation, we conduct a visualization study on a representative sample from the code task of LongBench. We generate the first output token using DeepSeek-V3.2 and extract the full attention distributions at each layer. We visualize the attention weights over all context tokens as a 2D heatmap, where the x-axis denotes token positions and the y-axis denotes layer indices.

The visualization reveals a pattern: tokens with high attention weights tend to form contiguous spans rather than appearing as isolated points in a considerable number of tasks. These high-density regions often correspond to semantically coherent segments (e.g., code blocks, mathematical formulas and derivations) and persist across multiple layers. Outside these spans, attention scores are negligible. This observation suggests that attention mass may be naturally concentrated in block-wise regions. Therefore, block-level sparsification can retain most of the informative attention distribution while avoiding the fine-grained selection overhead of token-wise top-k methods. The results provide empirical support for the two-stage hierarchical structure of HISA.

5.6 Selection Consistency: IoU Analysis

To quantify how closely HISA recovers DSA’s token selections, we measure the Intersection-over-Union (IoU) between their selected token sets under the deployment configuration $B = 128, m = 64$:

$$\text{IoU}(\mathcal{T}_t^H, \mathcal{T}_t) = \frac{|\mathcal{T}_t^H \cap \mathcal{T}_t|}{|\mathcal{T}_t^H \cup \mathcal{T}_t|}. \quad (12)$$

The consistently high layer-wise overlap and the graceful position-wise decay in Table 3 are consistent with HISA’s LongBench accuracy staying close to DSA overall (Table 2) while occasionally diverging on the hardest, longest-context queries. HISA therefore preserves the same token-index output contract for Sparse MLA while approximating, rather than exactly reproducing, DSA’s selected token set. Appendix C further reports a failure-case analysis and a hyperparameter sensitivity study, including an ablation over substantially larger block sizes.

6 Conclusion and Future Directions

To address the emerging bottleneck caused by the $O(L^2)$ complexity of the DSA indexer, we propose HISA, a hierarchical indexing approach. Specifically, HISA first uses a hardware-friendly block indexer to efficiently filter out a large number of irrelevant tokens, and then applies token-level reranking over the remaining candidates to construct the token indices consumed by Sparse MLA. HISA delivers up to $3.75\times$ indexer-kernel speedup over

Table 3: Token-set IoU (Eq. 12) between HISA and DSA selected tokens on DeepSeek-V3.2 under the deployment configuration $B=128, m=64$, binned in two complementary ways. (a) Averaged over 10-layer bins across all 61 layers, IoU is uniformly high (92.58%–94.94%) with no systematic trend across depth, indicating that HISA’s coarse filter agrees with DSA’s token selection at every layer. (b) Averaged over 2K-token query-position bins, IoU starts at 96.91% for the earliest queries (2–4K) and declines monotonically to 78.66% at the longest evaluated positions (18.4–18.8K): as the prefix lengthens, more candidate blocks compete for the fixed top- m budget, so Stage 1 is more likely to disagree with DSA’s exhaustive token-level scoring on which blocks contain the true top- k tokens.

(a) Averaged over layers							
Layer range	0–9	10–19	20–29	30–39	40–49	50–59	60
IoU (%)	94.90	94.94	93.43	92.58	92.85	93.21	93.94

(b) Averaged over query token position					
Position range	2–4K	4–6K	6–8K	8–10K	10–12K
IoU (%)	96.91	94.74	93.83	91.88	88.09
Position range	12–14K	14–16K	16–18K	18–18.8K	
IoU (%)	85.07	86.47	79.28	78.66	

DSA at 64K context. As a plug-and-play module, HISA can directly replace the token indexer in DeepSeek-V3.2 and GLM-5. Without additional training, it closely matches DSA on LongBench and substantially outperforms the corresponding block-sparse baseline on NIAH. In a full SGLang serving stack, HISA improves throughput by $1.65\times$, TTFT by $1.70\times$, and TPOT by $1.55\times$ over the official DSA implementation.

Several avenues remain open: (1) *Reducing information loss in coarse filtering*: the current block-level stage represents each block with a single pooled vector, which can fail when a block crosses a semantic boundary and the pooled representation does not reflect the most important token. Potential mitigations include overlapping blocks, adaptive block boundaries, or replacing mean pooling with max pooling to better preserve salient outlier directions. (2) *Training-aware HISA*: while HISA currently operates as a training-free inference-time replacement, jointly training the block scoring stage may improve the coarse filter’s accuracy, particularly for such boundary cases. (3) *Serving optimization*: fusing the stages and tuning the newly introduced operators for small block top- m could further reduce the remaining serving overhead.

Acknowledgments

This work was partially supported by PKU Kunpeng & Ascend Center of Excellence and the CCF-Tencent Rhino-Bird Open Research Fund.

References

- Anthropic. Claude sonnet 4.6, 2026. URL <https://www.anthropic.com/news/claude-sonnet-4-6>.
- 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. *arXiv preprint arXiv:2308.14508*, 2024.
- Yushi Bai, Qian Dong, Ting Jiang, Xin Lv, Zhengxiao Du, Aohan Zeng, Jie Tang, and Juanzi Li. Indexcache: Accelerating sparse attention via cross-layer index reuse. *arXiv preprint arXiv:2603.12201*, 2026.
- Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.
- Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022.
- DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.
- DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.
- Qichen Fu, Minsik Cho, Thomas Merth, Sachin Mehta, Mohammad Rastegari, and Mahyar Najibi. Lazyllm: Dynamic token pruning for efficient long context LLM inference. *arXiv preprint arXiv:2407.14057*, 2024.
- GLM-5-Team. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.
- Google DeepMind. Introducing gemini 3, 2025. URL <https://blog.google/products-and-platforms/products/gemini/gemini-3-collection/>.
- 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.
- Greg Kamradt. Needle in a haystack — pressure testing llms. 2023. <https://github.com/gkamradt/LLMTest.NeedleInAHaystack>.
- Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *International Conference on Learning Representations*, 2025.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.
- Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top- p pruning. In *Advances in Neural Information Processing Systems*, volume 38, 2025.
- Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.
- Meta. The llama 4 model collection, 2025. URL <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>.
- MiniMax, Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, et al. MiniMax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025.

- Moonshot AI. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.
- Wentao Ni, Kangqi Zhang, Zhongming Yu, Oren Nelson, Mingu Lee, Hong Cai, Fatih Porikli, Jongryool Kim, Zhijian Liu, and Jishen Zhao. Double-p: Hierarchical top-p sparse attention for long-context LLMs. *arXiv preprint arXiv:2602.05191*, 2026.
- OpenAI. GPT-5.4: Openai’s most powerful model, 2026. URL <https://openai.com/index/gpt-5-4-thinking-system-card>.
- Matanel Oren, Michael Hassid, Nir Rosenfeld, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.
- Qwen. Qwen3.5: Native multimodal agentic model, 2026. URL <https://qwenlm.github.io/blog/qwen3.5>.
- Lei Wang, Yu Cheng, Yining Shi, Zhengju Tang, Zhiwen Mo, Wenhao Xie, Lingxiao Ma, Yuqing Xia, Jilong Xue, Fan Yang, and Zhi Yang. Tilelang: A composable tile-based programming model for ai systems. *arXiv preprint arXiv:2504.17577*, 2025.
- Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.
- Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489, 2016.
- 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.
- Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. Spargeattention: Accurate and training-free sparse attention accelerating any model inference. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.
- 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.
- Weilin Zhao, Zihan Zhou, Zhou Su, Chaojun Xiao, Yuxuan Li, Yanghao Li, Yudi Zhang, Weilun Zhao, Zhen Li, Yuxiang Huang, Ao Sun, Xu Han, and Zhiyuan Liu. Inflm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation. In *International Conference on Learning Representations*, 2026.

A Algorithm Pseudocode

Algorithm 1 provides the complete pseudocode for the HISA indexer.

Algorithm 1 HISA: Hierarchical Indexed Sparse Attention

Require: Query indexing representations $\{\mathbf{q}_{t,j}^I\}$, gating weights $\{w_{t,j}^I\}$, token indexing keys $\{\mathbf{k}_s^I\}_{s=1}^L$, block size B , block budget m , token budget k

Ensure: Selected token set $\mathcal{T}_t$ of size k

- 1: Partition prefix into $M = \lceil L/B \rceil$ blocks $\mathcal{B}_1, \dots, \mathcal{B}_M$
- 2: **for** $b = 1$ to M **do**
- 3: $\tilde{\mathbf{k}}_b^I \leftarrow \text{MeanPool}(\{\mathbf{k}_s^I \mid s \in \mathcal{B}_b\})$
- 4: **end for**
- 5: **for** each query position t **do**
- 6: *// Stage 1: Block-level coarse filter*
- 7: **for** $b = 1$ to M **such that** $\mathcal{B}_b$ precedes t **do**
- 8: $J_{t,b} \leftarrow \sum_j w_{t,j}^I \cdot \text{ReLU}(\mathbf{q}_{t,j}^I \cdot \tilde{\mathbf{k}}_b^I)$
- 9: **end for**
- 10: $\mathcal{C}_t \leftarrow \text{TopK}(J_{t,:}, m) \cup \{\text{first block, last block}\}$
- 11: $\Omega_t \leftarrow \bigcup_{b \in \mathcal{C}_t} \mathcal{B}_b$
- 12: *// Stage 2: Token-level refinement*
- 13: **for** $s \in \Omega_t$ **do**
- 14: $I_{t,s} \leftarrow \sum_{j=1}^{H^I} w_{t,j}^I \cdot \text{ReLU}(\mathbf{q}_{t,j}^I \cdot \mathbf{k}_s^I)$
- 15: **end for**
- 16: $\mathcal{T}_t \leftarrow \text{TopK}(\{I_{t,s} \mid s \in \Omega_t\}, k)$
- 17: **end for**
- 18: **return** $\mathcal{T}_t$

B Experimental Settings

We detail additional experimental settings in this section. All long-context evaluations were conducted in a **zero-shot** setting.

B.1 End-to-End Serving Settings

We evaluate end-to-end serving (Section 5.2) on DeepSeek-V3.2 with $8 \times$ NVIDIA H200 GPUs and tensor parallelism 8. The workload consists of 10 random requests, each with 128K input tokens and 512 generated tokens; all numbers are measured after JIT warm-up.

B.2 End-to-End Serving: Why SGLang

We measure end-to-end serving on SGLang rather than vLLM because SGLang exposes an explicit request-to-token-pool abstraction that cleanly accommodates HISA’s heterogeneous KV layout: the original token-level KV cache alongside HISA’s Stage-1 block-pooled key cache, which stores one pooled indexing key per block rather than per token. vLLM’s cache classes, in contrast, enforce a uniform page size across cache classes, which conflicts with this block-pooled cache having a different granularity from the token-level cache. Since this changes the serving stack used to measure speed relative to the quality evaluations in Sections 5.3 and 5.4 (both on vLLM), Appendix C.2 reports a LongBench cross-check confirming that HISA’s quality conclusions transfer between the two stacks.

B.3 Long-context Benchmarks

We evaluated the long-context performance using the Needle In A Haystack (NIAH) test and the LongBench benchmark. We tested two models: **DeepSeek-V3.2** and **GLM-5**. Both models were deployed using the vLLM online serving framework with **FP8** precision.

Table 4: End-to-end speedup of HISA ($B=128, m=64$) over the official DSA baseline in SGLang across request concurrencies 1, 2, and 4, under the same DeepSeek-V3.2, $8 \times H200$ (TP=8), 128K-input / 512-output workload as Table 1. TTFT speedup stays essentially flat at $1.75\text{--}1.76\times$ across all concurrencies, since HISA’s reduction in first-token indexing cost is a per-request property largely independent of batching. TPOT speedup instead grows markedly with concurrency, from $1.05\times$ at concurrency 1 to $1.55\times$ at concurrency 4: at low concurrency, per-step decoding latency is dominated by fixed serving overhead and low GPU occupancy that indexer speedup alone cannot offset, whereas higher concurrency exposes more of the decode-side indexing cost, making HISA’s saving more visible in TPOT. We report up to concurrency 4 because, under this 128K-context workload, the KV cache in our serving setup can hold at most four concurrent requests.

Concurrency	Throughput	E2E latency	TTFT	TPOT
1	$1.48\times$	$1.48\times$	$1.76\times$	$1.05\times$
2	$1.58\times$	$1.58\times$	$1.76\times$	$1.38\times$
4	$1.64\times$	$1.65\times$	$1.75\times$	$1.55\times$

NIAH Settings For the NIAH experiments, we utilized a customized evaluation codebase modified from the RULER² GitHub repository. We did not apply chat templates to either model to ensure a direct assessment of their raw retrieval capabilities.

LongBench Settings We evaluated LongBench using the lm-eval³ framework. The configurations for LongBench varied slightly depending on the model characteristics:

- **Chat Template Usage:** DeepSeek-V3.2 was evaluated with its standard chat template. In contrast, GLM-5 was evaluated *without* a chat template. This decision was made because using the template triggered an extended thinking process that exceeded the maximum generation length and significantly slowed down inference. Furthermore, disabling the thinking process while keeping the template resulted in inferior performance compared to not using the template at all.
- **Concurrency Settings:** The default number of concurrent requests (num.concurrent) was set to 20. However, due to Out-Of-Memory (OOM) issues specific to GLM-5 on certain tasks, we adjusted the concurrency: longbench_single was run with a concurrency of 1, and longbench_summary was run with a concurrency of 2.

Fairness of Comparison We emphasize that although the specific settings (e.g., concurrency, chat template) differ across models and tasks to accommodate their unique characteristics and hardware constraints, we ensure that the settings are **strictly aligned** when comparing different methods within the same model and task combination. This guarantees a fair and rigorous comparison.

C Additional Experimental Results

C.1 End-to-End Serving Concurrency

To assess whether HISA’s gains persist under batched serving, Table 4 extends the Section 5.2 evaluation to request concurrencies of 1, 2, and 4. The trends are consistent with a fixed per-request TTFT saving that becomes easier to realize in TPOT as batching increases GPU occupancy.

C.2 SGLang vs. vLLM Quality Consistency

Section 5.2 measures end-to-end serving speed on SGLang, while the quality evaluations in Sections 5.3 and 5.4 use vLLM. Since the HISA algorithm itself is unchanged across the two

²<https://github.com/NVIDIA/RULER>

³<https://github.com/EleutherAI/lm-evaluation-harness>

stacks, we re-run a subset of LongBench on SGLang to confirm that our quality conclusions carry over to the stack used for the E2E measurements.

Table 5: LongBench results for DeepSeek-V3.2 on the SGLang stack, alongside the corresponding vLLM averages from Table 2 for reference. The same qualitative trend holds on both stacks: DSA and HISA remain close to each other while both clearly outperform Block-Sparse.

Config	SQA	MQA	Sum	FS	Syn	Avg.	vLLM Ref.
DSA (official)	49.79	52.68	22.24	60.29	66.69	50.34	51.55
HISA ($B=128, m=64$)	49.26	51.82	22.07	59.75	67.33	50.05	51.14
Block-Sparse ($B=128, m=16$)	46.99	50.03	21.46	55.24	66.14	47.97	49.63

On SGLang, DSA (50.34) and HISA (50.05) remain close while both clearly outperform Block-Sparse (47.97), matching the vLLM trend reported in Table 2 (DSA 51.55, HISA 51.14, Block-Sparse 49.63). The two stacks are not expected to produce identical scores—they differ in scheduling, batching, and numerical details unrelated to HISA itself—but the ranking and relative gaps are consistent, supporting the use of SGLang for the E2E speed measurements in Section 5.2 without re-validating quality from scratch on that stack.

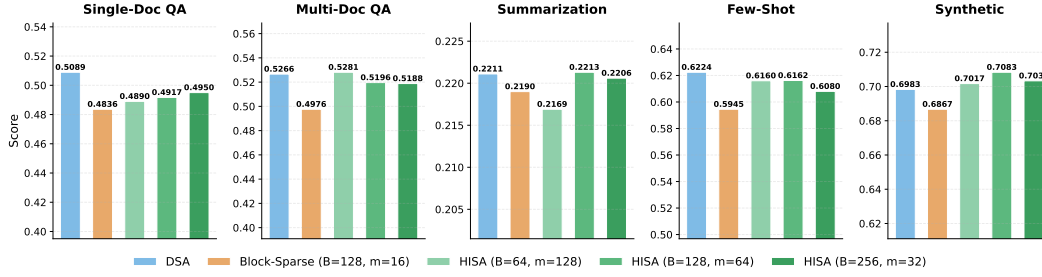
C.3 Hyperparameter Sensitivity

We investigate the sensitivity of HISA to its two key hyperparameters—block size B and block-level top- m —by comparing three HISA configurations that share the same candidate pool size, $mB = 8192$, but different coarse-to-fine trade-offs: ($B=64, m=128$), ($B=128, m=64$), and ($B=256, m=32$). We further include the original DSA as an upper bound and Block-Sparse ($B=128, m=16$) as a lower bound. All configurations use $k=2048$ for the final token selection. Results are evaluated on DeepSeek-V3.2 and GLM-5 across five LongBench task categories.

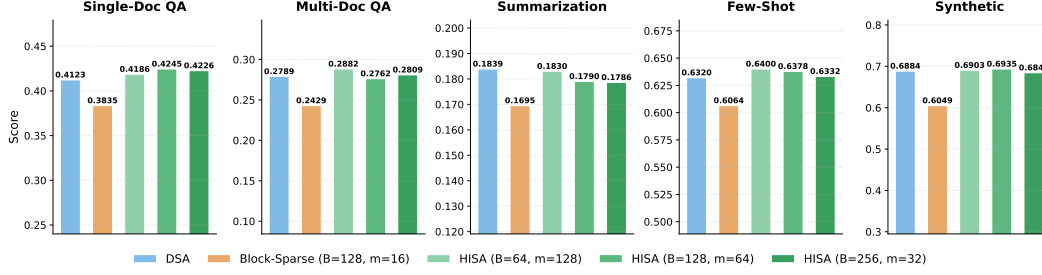
Figure 5 reveals two key findings. First, all three HISA configurations closely track DSA performance across all five task categories while consistently and substantially outperforming the Block-Sparse baseline. HISA is thus nearly lossless relative to the exhaustive flat scan, and token-level refinement is essential to this gap: even under the same block-level selection mechanism, the ability to prune low-relevance tokens *within* selected blocks—which Block-Sparse cannot do—yields measurable quality gains. Second, among the three HISA variants, the best block size is not uniform across sub-tasks and the trend in B is not monotonic: $B=64$ and $B=128$ tend to edge out $B=256$ overall, but the ranking is not identical across all task categories, and there is no single HISA configuration that is optimal everywhere. We attribute this non-monotonicity to the current training-free mean-pooling aggregation rather than to the hierarchical indexing framework itself; we analyze the underlying mechanism in detail below.

Why the trend is non-monotonic. Many of the per-task gaps in Figure 5 are small and partly within inference/evaluation variance, so nearby configurations should not be over-interpreted; we instead focus on the two categories that show the clearest, opposite trends on DeepSeek-V3.2—MQA favors smaller B , while SQA favors larger B . A smaller B is not uniformly better because, under training-free mean pooling, HISA’s coarse filter carries two interacting error sources:

- **Block-granularity error:** Stage 1 selects whole blocks rather than individual tokens, so smaller B is, in principle, always better for this term alone, since finer block granularity is strictly closer to DSA’s token-level selection. This error dominates for MQA, where evidence is fine-grained and scattered across multiple documents: under the fixed candidate budget mB , a larger B forces a smaller m , so fewer, coarser blocks are retained and scattered relevant fragments are more likely to be missed, whereas a smaller B with a correspondingly larger m gives finer granularity and broader coverage.



(a) Ablation study on DeepSeek-V3.2.



(b) Ablation study on GLM-5.

Figure 5: LongBench scores under different indexer configurations. All three HISA variants use a candidate token pool of size $mB = 8192$ and a final token budget of $k=2048$, with different choices of block size B and block-level top- m . The Block-Sparse baseline uses $B=128$ and $m=16$, corresponding to a candidate pool of 2048 tokens and no token-level refinement.

- **Block-representation error:** each block is summarized by a single mean-pooled vector; because this aggregation is training-free, the summary is not a particularly strong one, so its effect on B is not monotonic. For SQA, where the answer often lies in one long contiguous span, a moderately larger block can actually *improve* accuracy by pooling the whole span into a single, easily-retrieved block vector—so accuracy first rises above the small- B baseline, then turns over and declines once the block becomes so large that mean pooling blurs the useful span with irrelevant surrounding context.

We emphasize that the block-representation error is a consequence of training-free mean pooling, not of HISA’s hierarchical framework itself. With a trained block-level summarizer, this error should shrink and the block-granularity error would dominate more uniformly, so smaller B would more often improve accuracy as intuition suggests, restoring the more intuitive “smaller B is better” trend; we expect such a setting to find a stable B that matches DSA accuracy while preserving the hierarchical-indexing speedup (Section 6). We verify the current trade-off directly by evaluating substantially larger block sizes (up to $B=2048$) under the same fixed candidate budget $mB=8192$; see Appendix C.4 for the full results and category-level breakdown. In the current training-free setting, we recommend $B=128, m=64$ as a practical default: it stays close to DSA across all task categories (Table 2) and underlies the end-to-end speedups reported in Section 5.2, without requiring per-task hyperparameter search.

C.4 Extended Block-Size Ablation

Appendix C.3 attributes HISA’s non-monotonic block-size trend to two interacting error sources under training-free mean pooling: a *block-granularity error* that shrinks with smaller B , and a *block-representation error* from mean pooling whose dependence on B is not monotonic. To verify this trade-off directly, Table 6 extends the sweep to substantially larger blocks under the same fixed candidate budget $mB=8192$ on DeepSeek-V3.2—($B=512, m=16$),

($B=1024, m=8$), and ($B=2048, m=4$)—alongside the $B=128, m=64$ default from Table 2 for reference.

Table 6: LongBench results for DeepSeek-V3.2 with substantially larger block sizes, at the same fixed candidate budget $mB=8192$ used in Figure 5. The $B=128, m=64$ row is the deployment default (reproduced from Table 2 for reference); Avg. is computed over the same five task categories as Figure 5 (excluding Code).

Config	SQA	MQA	Sum	FS	Syn	Avg.
$B=128, m=64$ (default)	49.17	51.96	22.13	61.62	70.83	51.14
$B=512, m=16$	49.75	51.78	22.06	60.63	69.83	50.81
$B=1024, m=8$	49.66	51.07	22.04	60.45	69.83	50.61
$B=2048, m=4$	48.65	50.49	22.00	61.39	68.83	50.27

The results confirm the two-error explanation from Appendix C.3. For MQA, where evidence is fine-grained and scattered across multiple documents, the block-granularity error dominates and score declines steadily as B grows from 128 to 2048 ($51.96 \rightarrow 50.49$). For SQA, where the answer often lies in one long contiguous span, the block-representation error dominates instead, and the trend is non-monotonic rather than flat: accuracy first *rises* from $B=128$ to $B=512$ ($49.17 \rightarrow 49.75$), as a moderately larger block more effectively pools the whole evidence span into a single, easily-retrieved block vector, then turns over and declines through $B=1024$ (49.66) and $B=2048$ (48.65) once the block becomes large enough that mean pooling blurs the useful span with irrelevant surrounding context.

C.5 Failure Case Studies

We analyze cases where DSA is correct while HISA is not, to understand when and why the coarse filter can miss the correct evidence. Mechanistically, HISA can only lose information relative to DSA when Stage 1 fails to retain the block containing the relevant tokens: once a block survives coarse filtering, Stage 2 applies the identical DSA token-level indexer within it. The following three representative cases, with exact model outputs, trace this loss back to the current training-free mean-pooling instantiation: mean pooling can blur fine-grained entity relations, short local causal-action spans, or an isolated needle diluted by a much larger surrounding haystack, causing Stage 1 to assign a low score to the block that actually contains the answer.

Entity-relation blurring. A representative case from the LongBench 2WikiMQA subset asks: “When did William Le Poer Trench’s father die?” DSA answers correctly, “26 April 1872,” while HISA outputs a distractor date, “24 November 1837.” Answering requires first linking “William Le Poer Trench” to his father, “William Trench, 3rd Earl of Clancarty,” and then retrieving the father’s death date, while the surrounding context also names several other family members with similar titles. When the mean-pooled block representation blurs multiple overlapping entity relations together, the block containing the correct death date can receive a low Stage-1 score and never reach Stage 2.

Local causal-action dilution. A second case, from NarrativeQA, asks how slime gets into Dana’s apartment; the decisive evidence is a short local action span in which water from the faucet turns to slime and settles in the tub. DSA gives the precise answer, “through the pipes into the bathtub,” while HISA gives the more generic “through the apartment’s plumbing.” A short, relation-bearing action span can be outweighed at Stage 1 by nearby blocks that contain more salient but less precise lexical matches (e.g., “slime,” “bathtub,” “apartment”), biasing block selection away from the exact causal evidence.

Needle dilution. A third pattern is specific to Needle-in-a-Haystack: the relevant evidence there is an isolated needle whose signal occupies only a few tokens. When such a needle is pooled together with a much larger surrounding haystack, its signal can be dominated by the haystack in the mean-pooled block vector, so the block containing the needle may receive a low Stage-1 score.

All three patterns trace back to the same root cause: the current training-free mean pooling, not the hierarchical indexing framework itself—once a relevant block is retained, token-level accuracy is unaffected. Learned block-level representations in a trained HISA variant may better preserve such evidence while retaining the two-stage efficiency benefit, which we leave to future work (Section 6).