ASIR: AN AGENT-STRUCTURE-AWARE INDEXER WITH BLOCK-LOCAL REUSE FOR NATIVE SPARSE ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context agents accumulate conversations, tool outputs, and files, increasing the cost of attention over the growing history. Native sparse attention uses an Indexer to select a fixed number of tokens for attention, but selection still scans the full context. Hierarchical Indexers narrow the search by selecting a prescribed number of blocks before scoring their tokens. However, their budgets overlook request-specific differences in evidence coverage, and they rescan every selected block at each decoding step. We present ASIR, an Agent-Structure-Aware Indexer with block-local candidate reuse. During prefill, ASIR combines native segment boundaries, request relevance, and spatial extent to allocate candidate capacity. During decoding, it retains historical candidates in some selected blocks and refreshes others, scoring all candidates with the current query. The two components reduce the spatial extent and temporal repetition of Indexer scoring, respectively. On LongBench and LongMemEval, ASIR achieves aggregate quality comparable to a hierarchical baseline searching one quarter of the context while reducing both candidate-search capacity and repeated token-level scoring. Across 16K–128K contexts, ASIR achieves 2.23–$4.39\times$ Indexer speedup over native sparse attention and 1.11–$1.59\times$ over this hierarchical baseline.

1 INTRODUCTION

Agent execution continually accumulates instructions, dialogue turns, tool outputs, and file contents into contexts containing tens or hundreds of thousands of tokens. Dense attention accesses the growing KV cache at every decoding step, making long-context generation increasingly expensive. Native sparse attention, as instantiated by the DeepSeek Indexer-based design (DeepSeek-AI, 2025), addresses this cost by separating token selection from attention computation. A lightweight Indexer first computes relevance scores for historical tokens and selects a fixed number K; the attention operator then processes only the selected KV entries. This keeps the number of tokens used by sparse attention fixed even as the context grows. However, the Indexer still scores the complete prefix to identify those tokens. Figure 1(a) shows that Indexer selection accounts for an increasing share of traced decoding-kernel time as context length grows, making the Indexer itself an increasingly important optimization target.

Existing approaches reduce Indexer overhead through hierarchical search, computation reduction, and prediction or execution overlap (Xu et al., 2026; Zhou et al., 2026; Fu et al., 2026). HISA (Xu et al., 2026), a representative hierarchical Indexer, partitions the context into fixed-size blocks, selects a fixed number using current-query block scores, and performs token-level scoring within the selected blocks. This reduces the search range, but its prescribed budget does not distinguish between requests of the same length whose relevant evidence is concentrated or widely distributed.

A second limitation concerns how candidates are renewed within the selected regions. Rescanning every selected block couples region selection with full token-level search inside that region. Temporal reuse methods demonstrate that historical selections can reduce repeated search (Yao et al., 2026; Wu et al., 2024). We focus on the spatial granularity of candidate renewal: a selected region can contribute historical token positions while other regions expose all their positions to fresh search.

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.

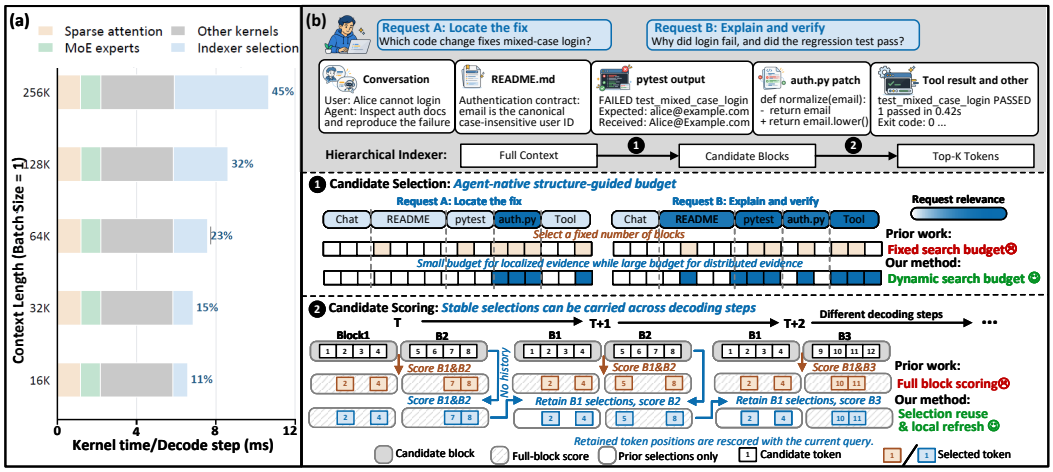

Figure 1: **The Indexer bottleneck and ASIR.** (a) Indexer selection occupies a growing share of traced decoding-kernel time over 16K–256K contexts; profiling details appear in Appendix A.6. (b) ASIR allocates structure-guided capacity and combines block-local candidate reuse with selective refresh. All candidates are scored with the current query. Selections are illustrative.

This separates region selection from candidate renewal, allowing full scans and historical candidate reuse to coexist within a decoding step. All resulting candidates are scored with the current query.

We observe two opportunities to address these limitations. First, sessions, dialogue turns, tool interactions, and files form native segments whose boundaries align with the spatial distribution of Indexer relevance. The extent of relevant information also varies across requests, suggesting that input structure can inform how much context a request needs to search. Second, token selections within a block often overlap across adjacent decoding steps. A block can therefore remain worth searching without requiring all of its token candidates to be rediscovered at every step. Yet retaining only historical positions would exclude tokens that become relevant later. This suggests treating candidate maintenance as a local update process: preserve useful historical positions while reopening selected regions to fresh search.

We propose ASIR, an Agent-Structure-Aware Indexer with Block-Local Reuse for native sparse attention. As illustrated in Figure 1(b), ASIR combines two complementary components. **Agent-Native Structure-Guided Candidate Capacity** estimates retrieval demand from request-conditioned segment relevance during prefill and sets a request-specific candidate budget for subsequent decoding. **Route-Conditioned Block-Local Candidate Reuse** makes token-candidate renewal a separate decision within the selected search range. Reused blocks retain previously selected positions, while refreshed blocks expose their eligible tokens to fresh search. Newly selected blocks are scanned, and retained blocks are periodically revisited. All candidate positions are scored with the current query. ASIR thus uses input structure to allocate search capacity and the retrieval hierarchy to maintain token candidates through local updates.

Our contributions are as follows:

- We propose **Agent-Native Structure-Guided Candidate Capacity**, which estimates request-level retrieval demand during prefill and uses it to allocate candidate capacity for subsequent decoding, reducing unnecessary token-level Indexer scoring.
- We propose **Route-Conditioned Block-Local Candidate Reuse**, which separates the selection of search regions from the renewal of their token candidates. It maintains candidates through local reuse and periodic block refresh, allowing fresh token discovery in part of the search range while retaining historical positions elsewhere.
- We evaluate ASIR on LongBench (Bai et al., 2023) and LongMemEval (Wu et al., 2025). ASIR retains aggregate quality comparable to HISA-$L/4$, while operator-level analysis shows up to $4.39\times$ speedup over native sparse attention and $1.59\times$ over HISA-$L/4$. Boundary diagnostics and refresh ablations examine the structural signal and the quality–efficiency trade-off.

2 RELATED WORK

2.1 INDEXER ACCELERATION AND ADAPTIVE SEARCH

Long-context sparse-attention work explores fixed or content-dependent sparsity, routing, and execution optimization (Child et al., 2019; Kitaev et al., 2020; Beltagy et al., 2020; Zaheer et al., 2020; Roy et al., 2021; Choromanski et al., 2021; Dao et al., 2022; Ding et al., 2024; Liu et al., 2024). ASIR focuses on the learned Indexer that constructs the final sparse-attention support.

HISA (Xu et al., 2026) accelerates the learned Indexer through coarse-to-fine search. It partitions the prefix into fixed-size blocks, constructs a representative for each block by pooling its indexing keys, and uses the current query to select the top-m blocks. The native token-level Indexer then scores all tokens within the selected blocks and produces the final support. This reduces the search range while retaining fine-grained token selection. However, the candidate capacity follows a prescribed budget or compression ratio and therefore does not distinguish requests whose relevant evidence is concentrated from requests whose evidence is distributed. The refinement stage also rescans all tokens in the selected blocks at every decoding step. MISA (Zhou et al., 2026) explores a complementary optimization by activating a query-dependent subset of Indexer heads to reduce token-scoring computation.

Other methods reduce long-context access through query-dependent routing, adaptive sparsity, or execution overlap. Quest (Tang et al., 2024) estimates page relevance from the current query and page-level key statistics. Double-P (Ni et al., 2026) adapts retrieval and attention computation through hierarchical top-p selection. MInference (Jiang et al., 2024) and SparDA (Fu et al., 2026) further study dynamic sparse retrieval or prediction-based overlap in related long-context settings. These methods reduce access or execution cost, but they do not use the native structure of an Agent context to estimate the candidate capacity required by a request.

ASIR retains regular blocks as computation units but uses Agent-native segments to estimate request-level search demand. Unlike HISA's prescribed capacity, its budget is derived from request-conditioned segment coverage during prefill. Block relevance selects promising regions at the current step, while the structural prior determines how broadly the request should search.

KV-cache compression and selective-attention methods address a different part of the pipeline. StreamingLLM, H2O, Scissorhands, SnapKV, PyramidKV, and DuoAttention reduce the amount of historical state retained or attended to (Xiao et al., 2024; Zhang et al., 2023; Liu et al., 2023; Li et al., 2025; Cai et al., 2025; Xiao et al., 2025). Their primary objective is to reduce the final KV support or its memory cost, whereas ASIR targets the cost of constructing token candidates before that support is formed.

2.2 TEMPORAL REUSE AND LOCAL CANDIDATE REFRESH

Temporal reuse methods reduce repeated candidate discovery across decoding steps by carrying information from previous selections. ReTopK (Yao et al., 2026) maintains historical query–support pairs and retrieves supports associated with similar queries. It merges the retrieved positions with a recent token window and reranks the resulting candidates using the current query. Low query similarity triggers broader selection, while periodic exact refreshes limit error accumulation. TokenSelect (Wu et al., 2024) similarly caches selection results across similar queries. Other approaches reduce the frequency of importance-predictor updates (Akhauri et al., 2025) or combine speculative retrieval with correction (Liu et al., 2026).

These methods primarily determine which historical supports can be reused or when a selection mechanism should be updated. ASIR focuses on a different renewal dimension: the spatial granularity of fresh candidate discovery within a hierarchical search route. The current query first determines the block route, which specifies where candidate search is allowed. A separate block-local refresh policy then determines which selected blocks expose all eligible positions to fresh search and which blocks contribute their historical candidates.

Consequently, a selected block can remain in the current route without requiring a full scan to rediscover its token candidates, while other selected blocks admit positions that were absent from the previous selection. Reuse and fresh discovery therefore coexist within one decoding step, and

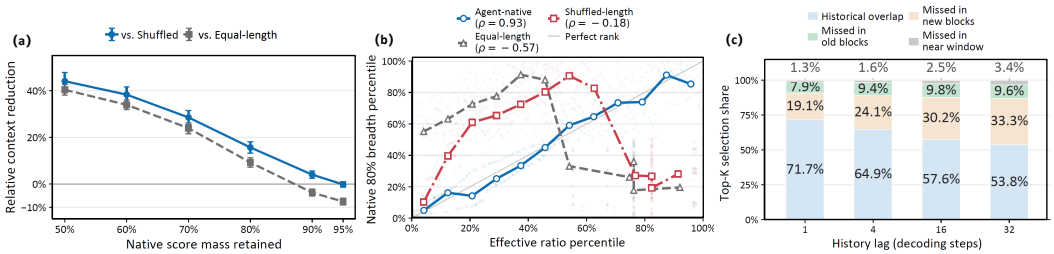

Figure 2: **Motivating observations.** (a) Relative context reduction from Agent-native boundaries against equal-length and shuffled-length partitions at matched native score-mass coverage. (b) Request-conditioned relevance dispersion versus native evidence breadth, with both axes shown as percentile ranks. (c) Historical overlap and uncovered-position categories across history lags. Measurement details appear in Appendix A.1.

all resulting positions are rescored with the current query. The distinction is orthogonal in spirit to query-based historical-support retrieval: a support retrieval method can determine which historical positions to reuse, while ASIR determines where new candidate discovery takes place within the current route.

3 METHOD

ASIR combines structure-guided capacity allocation with block-local candidate maintenance. Section 3.1 fixes the request-level capacity during prefill, Section 3.2 performs local reuse and refresh during decoding, and Section 3.3 summarizes the complete procedure.

3.1 AGENT-NATIVE STRUCTURE-GUIDED CANDIDATE CAPACITY

This component uses Agent-native context structure as a signal for allocating candidate capacity across requests. The signal is computed during prefill and fixes the search budget used by subsequent hierarchical retrieval. We describe the computation for one Indexer layer and omit layer indices. Let L denote the input context length and K the fixed number of tokens selected for sparse attention.

Agent contexts are naturally organized into dialogue turns, tool interactions, and files. Figure 2(a) shows that these native boundaries capture useful structure in the spatial distribution of native Indexer relevance. At moderate score-mass coverage targets, selecting complete Agent-native segments requires less context than equal-length or shuffled-length partitions when segments in each partition are ranked by their native reference score-mass density. The advantage becomes smaller as the target approaches complete coverage.

Figure 2(b) examines whether the dispersion of request-conditioned segment relevance is associated with the breadth of relevant evidence. We measure native evidence breadth as the fraction of regular blocks needed to capture a fixed proportion of native reference score mass. Across the pooled context–layer observations, a more dispersed Agent-native relevance signal is associated with broader evidence coverage, whereas the equal-length and shuffled-length controls do not show the same positive association.

These observations motivate combining request relevance with spatial extent when allocating candidate capacity. Segment relevance describes how strongly a region relates to the request, while segment extent describes how much context that region occupies. ASIR combines these properties into a block-level coverage signal and maps it to a request-specific candidate budget.

We first measure how strongly each segment relates to the original request. During prefill, request-token queries are compared with pooled key representatives formed from the tokens in each Agent-native segment. We apply the native Indexer scoring function to these query–representative pairs and aggregate the scores over request tokens, producing a segment relevance score $s_Q(u)$. We then normalize the segment scores into the request-level weights $w_Q(u)$ used below. The exact aggrega-

tion and normalization rules are given in Appendix A.1. The resulting weights form a request-level prior over the segments and are computed once before decoding.

To incorporate spatial extent, we partition the input into M regular blocks of nominal size B_{blk}. For each block p, let $u^\star(p)$ be its intersecting segment with the largest request relevance. We approximate the block's demand by weighting this relevance by the fraction of the block covered by that segment:

$$d_Q(p) = w_Q(u^\star(p)) \, \frac{|u^\star(p) \cap p|}{B_{\mathrm{blk}}}.$$

This approximation represents each block by its highest-relevance intersecting segment and scales that relevance by the segment's coverage within the block. A relevant segment spanning multiple blocks therefore contributes across its spatial extent, while partial coverage contributes proportionally less. Contributions from other intersecting segments are omitted.

We normalize these demands and measure their effective coverage:

$$\pi_p = \frac{d_Q(p)}{\sum_{p'} d_Q(p')}, \qquad \rho_Q = \frac{1}{M \sum_{p=1}^{M} \pi_p^2}.$$

If all demands are zero, we use a uniform distribution. The reciprocal concentration term $1/\sum_p \pi_p^2$ can be interpreted as the effective number of relevant blocks. Dividing by M gives $\rho_Q \in [1/M, 1]$: concentrated relevance produces a smaller value, while relevance spread across many blocks produces a larger value.

We treat the effective ratio as a relative coverage indicator and use linear interpolation to map it monotonically to a bounded candidate budget:

$$C = \min\left\{L, \, \lceil C_{\min} + \rho_Q (C_{\max} - C_{\min}) \rceil_{B_{\mathrm{blk}}}\right\}.$$

Here

$$\lceil x \rceil_{B_{\mathrm{blk}}} = B_{\mathrm{blk}} \left\lceil \frac{x}{B_{\mathrm{blk}}} \right\rceil$$

denotes rounding up to a multiple of the logical-block size. The outer minimum clips the rounded capacity to the available input length. The nominal endpoints satisfy $K \le C_{\min} < C_{\max} \le L$.

The endpoint values used in the main evaluation and the sensitivity comparison are specified in Section 4.1 and Appendix A.3, respectively. Thus, concentrated requests receive smaller search budgets, whereas requests with broader relevance coverage receive larger ones.

For each request, we compute ρ_Q and C once per Indexer layer and keep them fixed throughout decoding. At each step, the current Indexer query still scores block representatives and selects the current block route under this fixed capacity. The structural signal controls how broadly the request searches, whereas the current query controls which regions are selected.

Let $\mathcal{B}_t$ denote the selected blocks at step t and let P_p denote the valid token positions in block p. Their union forms the token search pool

$$D_t = \bigcup_{p \in \mathcal{B}_t} P_p.$$

Without temporal reuse, the token-refinement stage scores every position in D_t using the native Indexer scoring function and selects the final K tokens. Section 3.2 reduces this repeated scoring through block-local candidate reuse and selective refresh.

3.2 Route-Conditioned Block-Local Candidate Reuse

Selecting a block identifies a region worth searching, but does not require rediscovering every token candidate in that region at every step. Figure 2(c) shows that historical Top-K positions cover a substantial fraction of the current reference selection. Uncovered positions occur both in blocks absent from the historical route and in blocks already present at that historical step. Thus, retaining a block in the route does not imply that its historical candidate set remains complete.

We therefore separate region selection from candidate renewal. The current block route specifies where candidate search is allowed, while a block-local refresh policy specifies how each selected region supplies candidates. Refreshed blocks expose all eligible positions to fresh search, while the remaining blocks contribute their historical positions. This allows a region to remain selected without requiring a full scan of its tokens, while other regions admit previously omitted candidates.

Let D_t be the current token search pool from Section 3.1, $\mathcal{B}_t$ its block route, and T_{t-1} the token positions selected at the previous step in the same layer. The reusable history for block p is

$$H_{t,p} = T_{t-1} \cap D_t \cap P_p.$$

Intersecting history with D_t removes positions from blocks that have left the current route. Historical candidates are therefore retained only when their regions remain selected by the current query.

Let $\mathcal{R}_t \subseteq \mathcal{B}_t$ denote the blocks refreshed at the current step. Refreshed blocks contribute all their current positions, allowing previously unselected tokens to compete; the remaining blocks contribute only their historical candidates:

$$\widetilde{\mathcal{C}}_t = \underbrace{\bigcup_{p \in \mathcal{R}_t} (D_t \cap P_p)}_{\text{fresh candidate discovery}} \cup \underbrace{\bigcup_{p \in \mathcal{B}_t \setminus \mathcal{R}_t} H_{t,p}}_{\text{historical candidate retention}} .$$

Let p_t denote the logical block containing the most recently available context position. Let W_{near} denote the width of the recent logical-block window. The window contains the current logical block and the preceding $W_{\text{near}} - 1$ logical blocks:

$$\mathcal{W}_t = \{p \in \mathcal{B}_t : p_t - W_{\text{near}} + 1 \le p \le p_t\}.$$

Blocks in $\mathcal{W}_t$, newly routed blocks, and routed blocks without usable history are refreshed immediately.

For the remaining historical blocks, we divide the logical block indices into even and odd groups. At every N-th decoding step, one of the two groups is refreshed, and the group alternates at successive refresh events. Thus, each parity group is revisited once every $2N$ decoding steps. This schedule provides recurring opportunities for fresh discovery without requiring an additional token-level change detector. The resulting refresh set determines $\widetilde{\mathcal{C}}_t$; if the candidate pool is smaller than K, additional unrefreshed blocks are opened in route-ranking order.

All candidates, including reused positions, are scored using the current query:

$$T_t = \text{NativeIndexerTopK}\left(q_t^{\text{idx}}, \widetilde{\mathcal{C}}_t, K\right),$$

where q_t^{idx} is the Indexer query at step t. We reuse token positions rather than stale relevance scores. This separates the decision to reopen a region from the ranking of candidates within the resulting pool, reducing repeated token-level scoring while retaining the native scoring function and final attention budget. Selective refresh mitigates omissions, but does not guarantee the same support as rescanning the complete route at every step.

3.3 Algorithm

Algorithm 1 summarizes the per-layer procedure: prefill fixes the request-level capacity, while each decoding step updates the block route, combines historical and refreshed positions, and applies native Top-K scoring.

Algorithm 1 ASIR: Structure-Guided Capacity and Block-Local Reuse

Require: Input context X of length L; request Q; native segments $\mathcal{U}$
Require: Native Indexer queries and keys; final token budget K; block size B_{blk}
Require: Capacity endpoints $C_{\min}, C_{\max}$; recent-window definition and periodic refresh schedule
Ensure: Selected token positions T_t at each decoding step for one Indexer layer
Prefill: allocate request-level capacity
1: Partition X into regular blocks $\mathcal{P}$

Compute segment relevance w_Q from request queries and pooled segment key representatives
for all $p \in \mathcal{P}$ **do**
    $u^\star(p) \leftarrow$ highest-relevance segment intersecting p
    $d_Q(p) \leftarrow w_Q(u^\star(p))\,|u^\star(p) \cap p|/B_{\mathrm{blk}}$ ▷ relevance and coverage
end for
Normalize $\{d_Q(p)\}$ into $\boldsymbol{\pi}$ ▷ uniform if all demands are zero
$\rho_Q \leftarrow \left(|\mathcal{P}| \sum_{p\in\mathcal{P}} \pi_p^2\right)^{-1}$
$C \leftarrow \min\left\{L,\ \lceil C_{\min} + \rho_Q(C_{\max} - C_{\min})\rceil_{B_{\mathrm{blk}}}\right\}$ ▷ fixed throughout decoding
Decode: update the route and maintain candidates locally
$T_0 \leftarrow \varnothing$; $\mathcal{B}_0 \leftarrow \varnothing$
for each decoding step t **do**
    Update block metadata for the current prefix ▷ include newly generated positions
    Score block representatives using q_t^{idx}
    Select the current block route $\mathcal{B}_t$ under capacity C
    $D_t \leftarrow \bigcup_{p\in\mathcal{B}_t} P_p$ ▷ P_p: valid positions in block p
    $H_{t,p} \leftarrow T_{t-1} \cap D_t \cap P_p$ for each $p \in \mathcal{B}_t$
    $\mathcal{R}_t \leftarrow \{p \in \mathcal{B}_t : p$ intersects the recent window $\vee\ p \notin \mathcal{B}_{t-1} \vee H_{t,p} = \varnothing\}$
    Add periodically scheduled blocks from $\mathcal{B}_t$ to $\mathcal{R}_t$ ▷ revisit previously selected regions
    $\widetilde{\mathcal{C}}_t \leftarrow \varnothing$
    for all $p \in \mathcal{B}_t$ **do**
        if $p \in \mathcal{R}_t$ **then**
            $\widetilde{\mathcal{C}}_t \leftarrow \widetilde{\mathcal{C}}_t \cup (D_t \cap P_p)$ ▷ discover candidates
        else
            $\widetilde{\mathcal{C}}_t \leftarrow \widetilde{\mathcal{C}}_t \cup H_{t,p}$ ▷ retain historical positions
        end if
    end for
    while $|\widetilde{\mathcal{C}}_t| < K$ **do**
        Select an unrefreshed block $p \in \mathcal{B}_t \setminus \mathcal{R}_t$ in route-ranking order
        $\mathcal{R}_t \leftarrow \mathcal{R}_t \cup \{p\}$; $\widetilde{\mathcal{C}}_t \leftarrow \widetilde{\mathcal{C}}_t \cup (D_t \cap P_p)$ ▷ ensure sufficient candidates
    end while
    $T_t \leftarrow \mathrm{NativeIndexerTopK}(q_t^{\mathrm{idx}}, \widetilde{\mathcal{C}}_t, K)$ ▷ score all candidates with the current query
    Retain T_t and $\mathcal{B}_t$ as history for the next step
end for

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets and model. We evaluate ASIR with Keye-VL-2.0-30B-A3B (Kwai Keye Team et al., 2025) on text-only tasks from LongBench (Bai et al., 2023) and LongMemEval (Wu et al., 2025). We use fixed long-context subsets selected by the input-length protocol described in Appendix A.2. LongMemEval evaluates ASIR using the conversation structure provided by the benchmark. LongBench extends the evaluation to heterogeneous long contexts, using boundaries constructed from document, dialogue, and code structure as described in the same appendix. All methods use identical model weights, tokenizer, prompts, and paired input contexts.

Baselines and methods. We compare native sparse attention with HISA-8192, which uses a fixed candidate capacity of 8192 tokens, and HISA-$L/4$, which uses one quarter of the current prefix length. All hierarchical methods use logical blocks of size $B_{\mathrm{blk}} = 128$, matching HISA. ASIR-C uses only Agent-Native Structure-Guided Candidate Capacity. Full ASIR additionally uses Block-Local Candidate Reuse. For the main ASIR configuration, the recent window contains four logical blocks, the alternating refresh interval is $N = 4$, and the first periodic refresh updates odd-indexed logical blocks. The main capacity controller uses $\alpha_{\min} = 3/16$ and $\alpha_{\max} = 1/4$; endpoint sensitivity is reported in Appendix A.3.

Decoding and quality evaluation. All methods use greedy decoding with an output limit of 1024 tokens, EOS stopping, and final sparse-attention support size $K = 2048$. LongBench uses its official task-specific metrics; we report the macro-average over six task categories. LongMemEval uses a common question-type-specific judging protocol. Further details are provided in Appendix A.5.

Table 1: Task quality on paired LongBench and LongMemEval inputs. LongBench columns report category scores for single-question answering (SQA), multi-question answering (MQA), summarization (Sum), few-shot (FS), synthetic (Syn), and code (Code), together with their macro-average. LongMemEval reports judge accuracy in percent.

Method	SQA	MQA	Sum	FS	Syn	Code	LB Avg.	LME
Native	57.84	48.77	16.84	54.59	19.35	14.49	35.31	48.41
HISA-8192	57.69	45.36	17.73	56.42	18.28	16.83	35.39	45.22
HISA-$L/4$	53.38	47.88	17.34	51.99	17.20	16.27	34.01	46.38
ASIR-C	55.51	47.24	17.27	52.19	15.05	16.77	34.01	46.09
ASIR	57.65	46.30	17.40	51.86	15.05	16.54	34.14	46.38

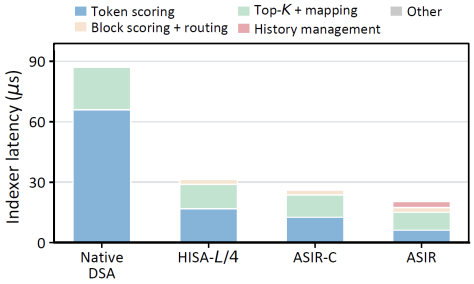

Figure 3: 64K Indexer stage latency on an NVIDIA H20 ($B_q = 1$). ASIR uses $N = 4$ and $K = 2048$.

Table 2: Native-relative Indexer speedup ($B_q = 1$).

Method	16K	32K	64K	128K
Native	1.00×	1.00×	1.00×	1.00×
HISA-8192	1.96×	2.46×	2.92×	3.06×
HISA-$L/4$	2.00×	2.46×	2.81×	2.76×
ASIR-C	2.27×	2.85×	3.38×	3.33×
ASIR	2.23×	3.10×	4.31×	4.39×

Indexer operator evaluation. We evaluate the complete decoding-time Indexer on a single NVIDIA H20 GPU at context lengths $L \in \{16\text{K}, 32\text{K}, 64\text{K}, 128\text{K}\}$ with request batch size $B_q = 1$. Each invocation processes one decoding query, including all Indexer heads, for one request at one layer. The measured scope includes block scoring and routing where applicable, candidate construction, token scoring, final Top-K selection and mapping, and method-specific history maintenance. All methods use the same query and context-key inputs and return $K = 2048$ selected token positions. Sparse attention and prefill-time capacity initialization are excluded. Appendix A.6 provides the timing protocol and signal-construction overhead.

4.2 Main Quality Results

Table 1 shows that ASIR remains close to HISA-$L/4$ in aggregate quality while reducing Indexer latency. ASIR-C obtains 34.01 on LongBench and 46.09% on LongMemEval, while adding block-local reuse yields 34.14 and 46.38%, respectively.

The category-level results show heterogeneous differences across task families rather than a uniform shift in one direction. ASIR remains competitive with HISA-$L/4$ across the evaluated benchmarks, while the magnitude and direction of differences vary across categories and question types. These labels describe different evidence organization and evaluation protocols, rather than a ranking of task difficulty.

This variation is consistent with the roles of ASIR's two components: structure-guided capacity adapts the search breadth to the request signal, while block-local reuse controls candidate renewal within selected regions. Their effects depend on evidence distribution, cross-region recovery, and the evaluation metric. Thus, the aggregate result reflects category-level trade-offs rather than a capability boundary tied to a particular task type. Detailed per-task and question-type results are reported in Appendix A.5.

4.3 Indexer Operator Efficiency

Table 2 reports complete Indexer speedups across context lengths. At 64K, ASIR-C provides a 1.20× speedup over HISA-$L/4$, and block-local reuse provides a further 1.28× speedup over ASIR-C. Figure 3 separates candidate processing from routing and history maintenance. At 16K, full ASIR

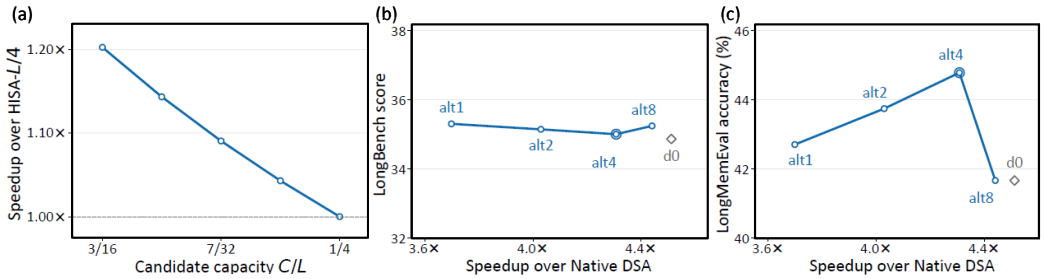

Figure 4: Capacity and refresh ablations. (a) Complete Indexer speedup over HISA-$L/4$ as candidate capacity varies with reuse disabled. (b) LongBench and (c) LongMemEval development quality versus Indexer speedup over native sparse attention. Operator timings use a single NVIDIA H20 GPU at 64K with $B_q = 1$ and $K = 2048$. Lines connect $N \in \{1, 2, 4, 8\}$; diamonds denote $d0$, and rings mark $N = 4$.

is slightly slower than ASIR-C, showing that reuse overhead can offset its savings at shorter context lengths. Appendix A.6 reports all refresh configurations.

4.4 Capacity and Refresh Ablations

Candidate capacity. Figure 4(a) examines candidate capacity with reuse disabled. Reducing C from $L/4$ to $3L/16$ increases the Indexer speedup over HISA-$L/4$ from $1.00\times$ to $1.20\times$. Token-scoring and final-selection work decrease with C, while routing remains fixed; the quality trade-off is reported in Appendix A.3.

Refresh interval. The two quality panels in Figure 4(b,c) use a common 64K operator reference. LongBench varies little across policies, whereas LongMemEval is more sensitive. We select $N = 4$ for the highest LongMemEval development accuracy, with efficiency close to $N = 8$; complete results are given in Appendix A.4.

Capacity-assignment control. We evaluate the sensitivity of request-conditioned capacity assignment with a rank-reversed control. The control uses the same set of capacity values as ASIR-C, but assigns them in reverse rank order within matched context-length strata, thereby breaking the original request-to-capacity correspondence while retaining the same routing, boundaries, decoding configuration, and candidate-reuse setting.

On 96 paired LongBench examples, ASIR-C obtains a category macro-average of 31.56, compared with 30.23 for the rank-reversed control ($+1.33$ points; task-stratified paired-bootstrap 95% CI $[+0.50, +2.29]$). ASIR-C is better on 27 examples, the control is better on 10, and 59 examples are tied. On 96 paired LongMemEval questions, both methods obtain 48.96% accuracy; the small changes are limited to two questions and cancel in aggregate.

The realized candidate work is close but not identical: the rank-reversed control uses 1.52% more candidate tokens on LongBench and 0.31% more on LongMemEval. We therefore interpret this experiment as a capacity-assignment sensitivity analysis rather than an exactly budget-matched causal test. The complete protocol, workload audit, and question-type breakdown are given in Appendix A.7.

5 Conclusion

We presented ASIR, which uses Agent-native structure for request-level Indexer capacity and selectively refreshes candidates within blocks. On LongBench and LongMemEval, ASIR maintains aggregate quality comparable to HISA-$L/4$ while achieving 2.23–$4.39\times$ speedup over native sparse attention and 1.11–$1.59\times$ over HISA-$L/4$. These results support structure-guided capacity allocation and local candidate maintenance as complementary ways to reduce Indexer overhead.

AI USE STATEMENT

We used generative AI tools to assist with manuscript drafting and revision, including suggestions on wording, grammar, and organization, and to generate icons used in Figure 1(b). AI tools also assisted with writing and debugging code for running experiments and processing and summarizing experimental outputs, and provided feedback on experimental protocols and the interpretation of results.

We additionally used AI agents to annotate and review LongBench segment boundaries according to predefined structural rules, followed by manual spot checks. The procedure is described in Appendix A.2. LongMemEval answers were evaluated using the LLM-based judging protocol described in Appendix A.5.

We have reviewed the AI-assisted text and figures, tested the AI-assisted code, and spot-checked the semantic-boundary annotations. We take responsibility for the final manuscript, code, annotations, and reported results.

REFERENCES

Yash Akhauri, Ahmed F. AbouElhamayed, Yifei Gao, Chi-Chih Chang, Sameh Gobriel, Nilesh Jain, and Mohamed S. Abdelfattah. Tokenbutler: Token importance is predictable. *arXiv preprint arXiv:2503.07518*, 2025. URL https://arxiv.org/abs/2503.07518.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023. URL https://arxiv.org/abs/2308.14508.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. URL https://arxiv.org/abs/2004.05150.

Zefan Cai, Yichi Zhang, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.02069.

Rewon Child, Scott Gray, Alec Paskalis, and Joel Hestness. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. URL https://arxiv.org/abs/1904.10509.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, et al. Rethinking attention with performers. *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2009.14794.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2205.14135.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Jian Ding, Li Ma, Shaogang Dong, et al. Longnet: Scaling transformers to 1,000,000,000 tokens. *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.02486.

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. URL https://arxiv.org/abs/2606.04511.

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, 2024. URL https://arxiv.org/abs/2407.02490.

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

Kwai Keye Team, Biao Yang, Bin Wen, Changyi Liu, Chenglong Chu, Chengru Song, Chongling Rao, Chuan Yi, Da Li, Dunju Zang, Fan Yang, Guorui Zhou, Hao Peng, Haojie Ding, Jiaming Huang, Jiangxia Cao, Jiankang Chen, Jingyun Hua, Jin Ouyang, Kaibing Chen, Kaiyu Jiang, Kaiyu Tang, Kun Gai, Shengnan Zhang, Siyang Mao, Sui Huang, Tianke Zhang, Tingting Gao, Wei Chen, Wei Yuan, Xiangyu Wu, Xiao Hu, Xingyu Lu, Yang Zhou, Yi-Fan Zhang, Yiping Yang, Yulong Chen, Zhenhua Wu, Zhenyu Li, Zhixin Ling, Ziming Li, Dehua Ma, Di Xu, Haixuan Gao, Hang Li, Jiawei Guo, Jing Wang, Lejian Ren, Muhao Wei, Qianqian Wang, Qigen Hu, Shiyao Wang, Tao Yu, Xinchen Luo, Yan Li, Yiming Liang, Yuhang Hu, Zeyi Lu, Zhuoran Yang, and Zixing Zhang. Kwai keye-vl technical report, 2025. URL https://arxiv.org/abs/2507.01949.

Yuhui Li, Yushi Huang, Bowen Yang, et al. Snapkv: Llm knows what you are looking for before generation. *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2404.14469.

Guangda Liu, Chengwei Li, Zhenyu Ning, Jing Lin, Yiwu Yao, Danning Ke, Minyi Guo, and Jieru Zhao. Freekv: Boosting kv cache retrieval for efficient llm inference. 2026. URL https://github.com/sjtu-zhao-lab/FreeKV.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.01889.

Zichang Liu, Jue Gao, et al. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.17118.

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. URL https://arxiv.org/abs/2602.05191.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021. doi: 10.1162/tacl_a_00353. URL https://arxiv.org/abs/2003.05997.

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

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.10813.

Wei Wu, Zhuoshi Pan, Kun Fu, Chao Wang, Liyi Chen, Yunchu Bai, Tianfu Wang, Zheng Wang, and Hui Xiong. Tokenselect: Efficient long-context inference and length extrapolation for llms via dynamic token-level kv cache selection. *arXiv preprint arXiv:2411.02886*, 2024. URL https://arxiv.org/abs/2411.02886.

Guangxuan Xiao, Yao Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.17453.

Guangxuan Xiao, Jue Gao, et al. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.10819.

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. *arXiv preprint arXiv:2603.28458*, 2026. URL https://arxiv.org/abs/2603.28458.

Wenshuai Yao, Wenyong Zhou, Hanyong Shao, Yizhe Chen, Zhiyuan Ning, Yuannuo Feng, Ru Huang, and Kechao Tang. Recall before you rank: Similarity-guided top-k reuse for efficient long-context attention. *arXiv preprint arXiv:2607.27692*, 2026. URL https://arxiv.org/abs/2607.27692.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2007.14062.

Zhenyu Zhang, Yuxin Sheng, Tianyi Tong, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.14048.

Ruijie Zhou, Fanxu Meng, Yufei Xu, Tongxuan Liu, Guangming Lu, Muhan Zhang, and Wenjie Pei. Misa: Mixture of indexer sparse attention for long-context llm inference. *arXiv preprint arXiv:2605.07363*, 2026. URL https://arxiv.org/abs/2605.07363.

A Additional Experimental Details and Results

A.1 Observation and Signal Protocols

Boundary alignment. For Figure 2(a), complete Agent-native segments are ranked by native reference score-mass density. Reference score mass is obtained by softmax normalization within the native reference Top-K positions. Relative context reduction is computed against the equal-length and shuffled-length partitions at the same retained score-mass target. For each partition and retention target, we first average the required context fraction across the evaluated layers and decode steps within each context. We then compute the relative reduction against each control for the same context. Error bars show normal 95% confidence intervals for the mean paired relative reduction, computed as $\bar{r} \pm 1.96\, s_r/\sqrt{n}$ across contexts. Layers and decode steps within a context are not treated as independent samples.

Request-conditioned signal. Figure 2(b) examines the association between request-conditioned relevance dispersion and native evidence breadth. For each segment u, we obtain its representative by mean pooling the token-level Indexer keys:

$$z_u = \frac{1}{|u|} \sum_{i \in u} k_i,$$

where k_i is the Indexer key at token position i. Let $a_{q,u} = \mathrm{IndexerScore}(q, z_u)$ denote the native Indexer score for request-token query q. We aggregate scores over the token span Q corresponding to the current user request or benchmark question:

$$s_Q(u) = \max\left\{0,\, \log\left(\frac{1}{|Q|} \sum_{q \in Q} \exp(a_{q,u})\right)\right\}.$$

This aggregation allows a strongly matching request token to contribute to segment relevance while accounting for the number of request tokens.

The implementation converts the aggregated segment scores into a normalized relevance distribution. Let $J = |\mathcal{U}|$ and define

$$\mu_Q = \frac{1}{J} \sum_{u \in \mathcal{U}} s_Q(u), \qquad \sigma_Q = \sqrt{\frac{1}{J} \sum_{u \in \mathcal{U}} \big(s_Q(u) - \mu_Q\big)^2}.$$

For $\sigma_Q > 10^{-12}$, the segment relevance weights are

$$w_Q(u) = \frac{\exp\big((s_Q(u) - \mu_Q)/\sigma_Q\big)}{\sum_{v \in \mathcal{U}} \exp\big((s_Q(v) - \mu_Q)/\sigma_Q\big)}.$$

Otherwise, we use the uniform distribution $w_Q(u) = 1/J$. The centering and scaling use the segment scores of the current request and introduce no learned parameters.

Generated tokens, model predictions, and decode-time queries are not used to construct w_Q. The segment-to-block demand mapping and effective-ratio capacity rule are defined in Section 3.1.

Native evidence breadth. Figure 2(b) compares the structure-guided effective ratio ρ_Q with an offline measure of native evidence breadth. For the latter, we sum native reference Top-K score mass within each regular block, rank blocks by this mass, and compute the minimum fraction of blocks needed to retain 80% of the reference mass.

The horizontal axis shows percentile ranks of ρ_Q, and the vertical axis shows percentile ranks of native evidence breadth. Lines connect binned medians, and Spearman correlations are computed over the pooled context–layer observations. Reference scores are used only to construct the offline breadth measure, not to compute ρ_Q or allocate capacity at runtime. The shuffled-length control permutes the original segment lengths and recomputes the structural signal under the resulting partition.

Temporal reuse coverage. Figure 2(c) uses reference traces collected across multiple contexts and Indexer layers under varying candidate capacities. At each step, the reference support is obtained by scoring all tokens in the selected block route with the native Indexer, with candidate reuse disabled. History lag denotes the number of decoding steps between the current and historical selections. Historical overlap is measured as the fraction of current reference Top-K positions also present in the historical selection.

Uncovered positions are assigned first to the recent window. The remaining positions are classified according to whether their blocks belonged to the historical route. These measurements describe potential reuse coverage; they do not measure the outcome of recursively applying the reuse and refresh policy.

A.2 Dataset Filtering and Boundary Construction

LongBench. We use a frozen long-context subset of LongBench selected by the tokenized input length. The retained inputs are at least 16K tokens long. Approximately 88.4% of the retained contexts fall in the 16K–32K range, while the remaining contexts fall in the 32K–64K range. For quality evaluation, we configure a total context budget of 128,000 tokens and reserve 1,024 tokens for generation, limiting input prompts to at most 126,976 tokens. This evaluation budget is below the checkpoint's configured maximum position length of 262,144 tokens. We preserve the original task names and task-specific evaluation metrics, and group the tasks into SQA, MQA, summarization, few-shot, synthetic, and code.

For LongBench, segment boundaries are constructed from the context alone, without using the question, reference answers, model predictions, or Indexer scores. The construction prioritizes explicit document and example separators, file and code-unit delimiters, speaker turns, section headings, and paragraph boundaries. Multi-document inputs preserve document separators; meeting and dialogue inputs follow speaker turns; repository inputs follow file and top-level code-unit boundaries; and few-shot inputs preserve example boundaries. Units longer than 4096 tokens are split into consecutive spans of at most 4096 tokens. The resulting segments are contiguous, non-overlapping, and cover the complete context. AI-assisted annotation followed the boundary rules above. The resulting annotations were reviewed by an additional AI agent for compliance with these rules, followed by manual spot checks by the authors. The annotations were frozen before evaluation.

LongMemEval. We preserve the conversation-session structure supplied by LongMemEval. Each serialized context retains its session and turn boundaries. The evaluated inputs satisfy the same 126,976-token input limit used for quality evaluation. The resulting contexts are concentrated around

100K–110K tokens and contain many conversation sessions, making session-level structure available to the hierarchical representation.

We retain each conversation session as a segment when its length does not exceed 4096 tokens. Longer sessions are partitioned at user–assistant round boundaries, grouping consecutive rounds into segments of at most 4096 tokens. A round that itself exceeds this limit is split into consecutive spans of at most 4096 tokens. Question-type labels, answer-session identifiers, and answer-related annotations are used only for evaluation and diagnostic analysis. They are not inputs to boundary construction or capacity allocation.

The same frozen contexts, boundaries, prompts, and generation settings are used for every method.

A.3 Capacity Endpoint Sensitivity

We compare two lower endpoints while keeping the upper endpoint fixed at $L/4$. Table 3 reports the resulting quality trade-off together with HISA-$L/4$, whose candidate capacity is length-scaled but request-independent.

Raising the lower endpoint from $5L/32$ to $3L/16$ increases the LongBench macro-average from 33.35 to 34.01, while LongMemEval accuracy decreases from 46.96% to 46.09%. We use $3L/16$ in the main experiments to provide a larger minimum candidate budget and match the LongBench macro-average of HISA-$L/4$. The two settings therefore represent different quality–capacity trade-offs; neither dominates on both benchmarks.

Table 3: Quality comparison for the lower capacity endpoint. LongBench reports the six-category macro-average and LongMemEval reports DeepSeek-judged accuracy.

Configuration	LongBench	LongMemEval (%)
HISA-$L/4$	34.01	46.38
ASIR-C, $C_{\min} = 5L/32$	33.35	46.96
ASIR-C, $C_{\min} = 3L/16$	34.01	46.09

A.4 Refresh-Frequency Development Results

We compare refresh configurations on fixed diagnostic subsets containing 96 LongBench examples and 96 LongMemEval questions. Examples are selected by round-robin sampling across task or question-type and input-length strata, with deterministic SHA-256 ordering of context identifiers within each stratum. Selection does not use reference answers or model scores.

The LongBench subset covers 16 tasks, with 82 inputs in $[16\text{K}, 32\text{K})$ and 14 in $[32\text{K}, 64\text{K})$. The LongMemEval subset contains 16 questions from each of the six question types. These subsets are drawn from the evaluation pools and are also included in the full-set results. All configurations use paired inputs and the quality protocol in Appendix A.5.

Table 4: Refresh-frequency development results on paired subsets.

Policy	LongBench score	LongMemEval accuracy (%)
d0	34.87	41.67
alt1	35.31	42.71
alt2	35.15	43.75
alt4	35.01	44.79
alt8	35.25	41.67

Quality does not vary monotonically with the refresh interval. The $N = 4$ configuration gives the highest LongMemEval development accuracy among the tested policies, while LongBench differences remain small. We use this configuration for the main evaluation.

Table 5: Per-task LongBench quality results using the official task-specific metrics.

Task	Native	HISA-8192	HISA-$L/4$	ASIR-C	ASIR
multifieldqa_en	100.00	100.00	100.00	100.00	100.00
dureader	33.50	33.26	30.65	34.03	24.45
passage_count	19.35	18.28	17.20	15.05	15.05
musique	41.61	42.46	41.84	41.89	41.70
vcsum	0.33	0.35	0.33	0.38	0.35
2wikimqa	60.00	46.67	60.00	53.33	60.00
qmsum	24.38	24.61	23.99	23.73	24.57
samsum	40.04	42.72	36.48	38.48	37.50
narrativeqa	35.71	35.29	34.63	35.00	35.16
lcc	18.80	21.00	22.00	22.60	20.40
repobench-p	10.18	12.66	10.55	10.95	12.68
hotpotqa	59.96	59.05	59.05	59.70	59.05
lsht	39.44	42.25	35.21	33.80	33.80
triviaqa	84.28	84.28	84.28	84.28	84.28
gov_report	25.82	28.22	27.69	27.71	27.29
qasper	37.80	37.80	25.53	31.55	37.80

Table 6: LongMemEval accuracy by question type. Values are percentages of judged-correct answers under the same fixed DeepSeek judge. The abstention row is an overlapping diagnostic subset. The ASIR column uses the $N = 4$ alternating-refresh configuration.

Question type	Native	HISA-8192	HISA-$L/4$	ASIR-C	ASIR
Knowledge update	66.04%	58.49%	62.26%	58.49%	58.49%
Multi-session	29.70%	26.73%	28.71%	27.72%	27.72%
Single-session assistant	90.00%	76.67%	83.33%	83.33%	86.67%
Single-session preference	9.52%	9.52%	4.76%	4.76%	4.76%
Single-session user	77.19%	78.95%	75.44%	78.95%	78.95%
Temporal reasoning	34.94%	33.73%	34.94%	34.94%	34.94%
Abstention diagnostic	16.00%	16.00%	16.00%	20.00%	20.00%
Overall	48.41%	45.22%	46.38%	46.09%	46.38%

A.5 Per-Task Quality and Scoring Protocol

LongBench. Table 5 reports LongBench scores using the official task-specific metrics. We average task scores within each category and then average the six category scores, giving each category equal weight regardless of its sample count. The effect of reuse varies across tasks. Relative to ASIR-C, ASIR improves Qasper from 31.55 to 37.80 and RepoBench-P from 10.95 to 12.68, while reducing DuReader from 34.03 to 24.45. The close aggregate scores therefore reflect offsetting task-level changes rather than uniform quality retention.

LongMemEval. Table 6 reports LongMemEval accuracy by question type. The six primary rows correspond to the benchmark's question types. The abstention row is an overlapping diagnostic subset, because abstention questions retain their original question type in the benchmark metadata. The overall value is computed over all evaluated questions and is therefore a micro-average rather than an unweighted average of the type-level values.

Scoring and judging. LongBench uses the original task-specific metrics. Exact match, F1, ROUGE, classification accuracy, and code-similarity metrics are computed within their respective tasks and are not mixed at the sample level.

For LongMemEval, every method uses the same question, frozen conversation context, generation configuration, and question-type- specific judging prompt. We use the fixed `deepseek-v4-pro` model as the judge with temperature 0 and reasoning disabled. Judge requests use a maximum output budget of 1024 tokens. The judge assigns a binary correctness decision according to the benchmark's answer-checking protocol. The reported overall LongMemEval value is the fraction of judged-correct answers over the complete evaluated set.

All quality comparisons use paired examples and greedy decoding with `max_new_tokens`=1024 and EOS stopping. Generated answers are retained together with their termination reason so that EOS termination and the maximum generation limit can be distinguished during analysis. The

Table 7: Complete Indexer latency and speedup on a single NVIDIA H20 GPU with $B_q = 1$. Each entry reports latency in microseconds, followed by speedup over native sparse attention in parentheses. ASIR uses $N = 4$.

Method	16K	32K	64K	128K
Native	35.10 (1.00×)	52.65 (1.00×)	87.75 (1.00×)	157.96 (1.00×)
HISA-8192	17.88 (1.96×)	21.41 (2.46×)	30.07 (2.92×)	51.54 (3.06×)
HISA-$L/4$	17.51 (2.00×)	21.41 (2.46×)	31.25 (2.81×)	57.24 (2.76×)
ASIR-C	15.50 (2.27×)	18.46 (2.85×)	25.99 (3.38×)	47.42 (3.33×)
ASIR-$d0$	15.47 (2.27×)	16.39 (3.21×)	19.45 (4.51×)	34.69 (4.55×)
ASIR-alt1	16.24 (2.16×)	18.53 (2.84×)	23.73 (3.70×)	42.40 (3.73×)
ASIR-alt2	16.01 (2.19×)	17.66 (2.98×)	21.78 (4.03×)	38.68 (4.08×)
ASIR-alt4 (ASIR)	15.72 (2.23×)	17.00 (3.10×)	20.37 (4.31×)	36.02 (4.39×)
ASIR-alt8	15.64 (2.24×)	16.92 (3.11×)	19.76 (4.44×)	34.92 (4.52×)

model-generation limit and the judge-response limit are separate: the former controls evaluated answers, while the latter only bounds the DeepSeek judge output.

A.6 Operator Timing Details

Motivating model profile. Figure 1(a) profiles native sparse attention in the full model using tensor parallelism of two and request batch size one. Frozen source contexts are repeated or truncated to obtain the target input lengths. We profile eight steady-state decoding steps with CUDA Graphs and report the additive kernel-time breakdown from the critical tensor-parallel rank. The 256K input contains 261,888 tokens, remaining within the checkpoint's configured position limit.

Execution and timing. We benchmark the complete Indexer on a single NVIDIA H20 with $B_q = 1$ at initial context lengths of 16K, 32K, 64K, and 128K tokens. Inputs are captured from native sparse attention runs on 16 fixed contexts, eight from each benchmark, at layers 0, 24, and 47. Context content is truncated or repeated to reach the target length while preserving the request text. Queries and keys use BF16, with FP32 score accumulation. All methods replay the same query and key sequence.

Each trial covers 65 consecutive decoding steps, starting with empty history. ASIR-C computes its capacity once from the initial input for each request and layer; all reuse configurations share this capacity. Routing and history states evolve sequentially during the replay.

We use CUDA Graph execution with 20 warm-up trials and 50 timed trials. CUDA events on the execution stream measure the complete sequence, including routing, candidate construction, token scoring, final Top-K selection and mapping, and history maintenance. Input preparation, capacity initialization, allocation, graph capture, and state reset are outside the timed region. Correctness is checked against each method's reference implementation before timing.

For each request–layer case, we divide sequence latency by 65 and take the median across timed trials. Reported latency is the arithmetic mean of these values across requests and layers; speedup is the ratio of the aggregated baseline and method latencies.

Complete operator results. Table 7 reports complete Indexer latency and speedup relative to native sparse attention across context lengths and refresh configurations. ASIR-C disables candidate reuse, while ASIR-$d0$ retains reuse without periodic remote refresh. ASIR-$\mathrm{alt}N$ alternates the refreshed historical block group every N decoding steps. The main ASIR configuration uses $N = 4$. All methods use the same final support size $K = 2048$.

Signal-construction cost and amortization. We benchmark the Signal-construction path on an NVIDIA H20 using synthetic fixed-shape inputs. The timed path includes segment pooling, request relevance, segment-to-block projection, and effective-ratio computation. We report the median CUDA-event latency over 101 graph replays, with metadata and workspace prepared before timing. The signal is constructed once during prefill and is not recomputed during decoding.

Table 8: Signal-construction latency and calculated amortization relative to HISA-$L/4$.

Context	GPU latency (μs)	Steps to amortize
16K	18.880	10
32K	22.848	8
64K	27.328	6
128K	39.616	5

Using the operator latencies in Table 7, we calculate the decoding steps required to offset this one-time GPU cost relative to HISA-$L/4$:

$$D_{\mathrm{GPU}} = \left\lceil \frac{T_{\mathrm{signal}}}{\bar{t}_{\text{HISA-}L/4} - \bar{t}_{\text{ASIR-}C}} \right\rceil,$$

where T_{signal} denotes signal-construction latency and $\bar{t}$ denotes mean Indexer latency per decoding step.

We compare against ASIR-C to isolate capacity adaptation from candidate reuse. The calculation uses unrounded latency values and assumes a positive, sustained mean per-step saving. Under the latency values reported here, the GPU signal-construction cost is amortized within 10 decoding steps.

A.7 Capacity-Assignment Sensitivity: Rank-Reversed Control

Purpose. The capacity controller assigns different candidate budgets to different requests according to their structure-guided retrieval signal. To examine whether the assignment itself matters, we compare ASIR-C with a rank-reversed control. Both arms use the same candidate capacity values, but the control assigns those values to requests in the opposite rank order within matched context-length strata.

The experiment disables candidate reuse in both arms. Consequently, the comparison isolates the capacity-assignment component while keeping the routing procedure, Agent-native boundaries, decoding configuration, and final sparse-attention support size fixed.

Paired evaluation protocol. We evaluate 96 paired LongBench examples and 96 paired LongMemEval questions. LongBench examples are stratified across tasks and context-length strata. LongMemEval questions are stratified across its six question types and context-length strata. The same input, prompt, model weights, tokenizer, generation limit, EOS policy, and judging procedure are used in the two arms.

For each request and profiled Indexer layer, ASIR-C first produces a capacity value $C_{i,\ell}$. Within each benchmark and matched context-length stratum, requests are ordered by their ASIR-C capacity fraction. The control assigns the capacity fraction from the opposite rank:

$$\widetilde{\rho}_{i,\ell} = \rho_{\pi_{\mathrm{rev}}(i),\ell}, \qquad \rho_{i,\ell} = \frac{C_{i,\ell}}{L_i},$$

where π_{rev} denotes the rank-reversing assignment. The resulting capacity is rounded upward to a multiple of the logical block size and clipped to the same valid interval as ASIR-C:

$$\widetilde{C}_{i,\ell} = \min\!\big(C^{\max}_{i,\ell}, \max\!\big(C^{\min}_{i,\ell}, \mathrm{round}_{128}(L_i \widetilde{\rho}_{i,\ell})\big)\big).$$

The capacity multiset is therefore preserved within each matched stratum, while its correspondence with the original request is reversed.

Quality results. Table 9 summarizes the paired quality comparison. LongBench values report the category macro-average and the task macro-average on the 96-example diagnostic cohort. LongMemEval values report the percentage of judged-correct answers on the 96-question cohort.

For LongBench, ASIR-C is better on 27 paired examples, the rank-reversed control is better on 10, and 59 examples are tied. A task-stratified paired bootstrap gives a 95% confidence interval of $[+0.50, +2.29]$ percentage points for the category-macro difference.

Table 9: Quality comparison for the rank-reversed capacity-assignment control.

Benchmark	n	ASIR-C	Rank-reversed	Difference
LongBench category macro	96	31.56	30.23	+1.33
LongBench task macro	96	37.49	36.01	+1.49
LongMemEval accuracy	96	48.96%	48.96%	0.00

Table 10: LongMemEval question-type results for the paired capacity-assignment comparison.

Question type	ASIR-C	Rank-reversed
Knowledge update	8/16 (50.00%)	9/16 (56.25%)
Multi-session	6/16 (37.50%)	5/16 (31.25%)
Single-session assistant	13/16 (81.25%)	13/16 (81.25%)
Single-session preference	1/16 (6.25%)	1/16 (6.25%)
Single-session user	13/16 (81.25%)	13/16 (81.25%)
Temporal reasoning	6/16 (37.50%)	6/16 (37.50%)
Overall	47/96 (48.96%)	47/96 (48.96%)

For LongMemEval, the two methods both obtain 47 correct answers out of 96. The only changes occur in two questions: ASIR-C is correct on one multi-session question on which the control is incorrect, while the control is correct on one knowledge-update question on which ASIR-C is incorrect.

LongMemEval question-type breakdown. The question-type results are shown in Table 10. Counts are reported as correct answers out of 16 questions per type.

Candidate-work audit. We audit realized candidate work on tensor-parallel rank 0 at layers 0, 12, 24, 36, and 47, beginning at decode step 2 and comparing only the common decode prefixes of the paired runs. The results are reported in Table 11.

Table 11: Realized candidate work in the rank-reversed capacity-assignment comparison.

Benchmark	ASIR-C mean candidates	Rank-reversed mean candidates	Relative change
LongBench	4919.17	4994.18	+1.52%
LongMemEval	20765.72	20829.23	+0.31%

Approximately 19.3% of the audited LongBench layer-step pairs and 32.7% of the audited LongMemEval layer-step pairs have identical capacities in the two arms. The mean relative capacity perturbation is 10.5% for LongBench and 1.8% for LongMemEval. All realized capacities remain within the valid $C_{\min}$–$C_{\max}$ interval, and the capacity-map consistency check reports no mismatches.

Interpretation. The rank-reversed control reduces LongBench quality while using a slightly larger average candidate budget, whereas the two methods tie on LongMemEval. These results show that the correspondence between the request-conditioned signal and the assigned capacity is relevant on the LongBench diagnostic cohort. Because the realized candidate work is not exactly identical, the comparison is interpreted as a capacity-assignment sensitivity analysis rather than a strictly budget-matched causal estimate of the structural signal.