TISA: Exploiting Temporal Locality to Accelerate Indexing Sparse Attention

Anonymous ACL submission

Abstract

Token-level sparse attention mechanisms such as DeepSeek Sparse Attention (DSA) rely on a lightweight indexer to score every prefix position and select the top-k tokens for downstream attention. While this design reduces the attention cost to $\mathcal{O}(Lk)$, the indexer itself retains $\mathcal{O}(L)$ per-query cost and becomes the dominant bottleneck at long contexts. Existing accelerations operate along spatial axes: sharing indices across layers (IndexCache), hierarchical search-space pruning (HISA), and indexer-head routing (MISA), each treating consecutive decode steps independently. We identify a complementary *temporal* axis: adjacent decode steps share over 90% of top-k selections at 4K context and maintain 95% stability among high-scoring core tokens even at 32K. Based on this observation, we propose TISA, which partitions prefix positions into K groups and refreshes only one group per step in round-robin fashion, reusing cached scores for the rest. TISA requires no retraining, no architectural modification, and no custom GPU kernel. Experiments on DeepSeek-V3.2 and GLM-5 show that TISA maintains competitive LongBench quality, preserves full Needle-in-a-Haystack accuracy at $K{=}2, 4, 8$ across contexts up to 128K, and achieves $3.8\times$ to $7.3\times$ indexer kernel speedup on NVIDIA H20 GPUs.

1 Introduction

As context windows of large language models scale from 128K toward 1M tokens (DeepSeek-AI, 2026; Qwen, 2025), the quadratic cost of self-attention becomes a dominant bottleneck in both prefill and decode. Token-level sparse attention offers an effective remedy: a lightweight *indexer* scores every prefix position, selects the top-k most relevant tokens, and routes only those to the full attention computation. This paradigm, introduced as DeepSeek Sparse Attention (DSA) (DeepSeek-AI, 2025) and adopted by GLM-5 (GLM-5-Team, 2026), achieves strictly finer-grained selection than block-level alternatives (Lu et al., 2025; Yuan et al., 2025) and enables $\mathcal{O}(Lk)$ attention at the cost of an $\mathcal{O}(L)$ indexer pass per query per layer.

However, as context length increases and the downstream attention becomes cheaper relative to selection, the indexer itself emerges as the new bottleneck, consuming up to 38% of total decode latency (Bai et al., 2026). Recent work has attacked this cost along three *spatial* axes: IndexCache (Bai et al., 2026) shares indices across layers, HISA (Xu et al., 2026) narrows the per-query search space via hierarchical filtering, and MISA (Zhou et al., 2026) routes queries to a subset of indexer heads. All three optimize *within* a single indexer invocation, treating consecutive decode steps as independent events.

We observe that they need not be treated independently. During autoregressive generation, the query vector evolves gradually: the newly generated token barely shifts the set of relevant prefix positions. Empirically, we find that adjacent decode steps share over 90% of their top-k selections at 4K context and still maintain 71% overlap at 32K (Section 3). Crucially, the instability concentrates at low-score boundary tokens: the top-128 highest-scoring positions exhibit 95% stability even at 32K, meaning the tokens that dominate the attention output rarely change between steps.

This temporal locality motivates a simple acceleration strategy. We propose **TISA** (**T**emporal **I**ndex **S**parse **A**ttention), which maintains a persistent score cache and refreshes only a fraction of it per step. Concretely, prefix positions are partitioned into K contiguous groups; at each decode step, one group is rescored using the current query while the remaining $K{-}1$ groups retain their cached scores. A round-robin schedule ensures every position is refreshed exactly once every K steps. The top-k selection always operates on the complete cache, so the downstream Sparse MLA

remains unmodified.

TISA opens a fourth optimization axis, **temporal reuse**, that is orthogonal to and composable with all three existing spatial approaches. It requires no retraining, no architectural change to the indexer, and no custom kernel: the same FP8 scoring kernel runs on a $1/K$-sized input slice. Experiments on DeepSeek-V3.2 and GLM-5 show that TISA maintains competitive LongBench quality, preserves full Needle-in-a-Haystack retrieval accuracy at $K{=}2, 4, 8$ across contexts up to 128K, and achieves $3.8\times$ to $7.3\times$ indexer kernel speedup on H20 GPUs, saving over 26 ms per token at 128K context across 61 layers. These results demonstrate the effectiveness of temporal reuse as a new optimization direction for indexing sparse attention.

2 Background

2.1 DeepSeek Sparse Attention

DSA (DeepSeek-AI, 2025) decomposes the attention computation into two stages: a lightweight *indexer* that selects a small token subset, followed by a *Sparse MLA* operator that computes exact attention only over the selected subset.

Indexer. Let L denote the causal prefix length for query position t. The indexer maintains per-token indexing keys $\mathbf{k}_s^I \in \mathbb{R}^{d_I}$ and, for each query, produces H^I indexing query vectors $\mathbf{q}_{t,j}^I \in \mathbb{R}^{d_I}$ with associated gating weights $w_{t,j}^I$. The relevance score aggregated over all indexer heads is:

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

The indexer then selects the k highest-scoring positions:

$$\mathcal{T}_t = \mathrm{TopK}\big(\{I_{t,s}\}_{s=1}^{L},\ k\big) . \qquad (2)$$

Sparse MLA. Given the selected set $\mathcal{T}_t$, the main attention operates in the Multi-head Latent Attention (MLA) mode (DeepSeek-AI, 2025), computing attention only over the latent KV entries at the selected positions:

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

where $\mathbf{c}_s$ is the compressed latent entry for token s. This reduces the per-layer attention cost from dense $\mathcal{O}(L^2)$ to sparse $\mathcal{O}(Lk)$.

Complexity bottleneck. While Sparse MLA scales favorably, the indexer must score every prefix token for every query at every layer, retaining $\mathcal{O}(L^2)$ per-layer complexity. As context lengths grow toward 128K tokens and beyond, the indexer transitions from a minor overhead to the dominant bottleneck, accounting for up to 38% of end-to-end decode latency (Bai et al., 2026).

2.2 Existing Optimization Axes

Recent work has proposed three directions to accelerate the DSA indexer, each reducing redundancy along a different *spatial* axis.

Cross-layer reuse (IndexCache). Bai et al. (2026) observe that top-k selections are highly similar across consecutive layers. IndexCache partitions layers into a small number of *Full* layers that run their own indexers and a majority of *Shared* layers that reuse the nearest Full layer's indices. The optimization axis is the **layer dimension**: fewer layers invoke the indexer.

Hierarchical search (HISA). Xu et al. (2026) replace the flat full-prefix token scan with a two-stage coarse-to-fine search. A block-level stage first scores pooled block representatives to prune irrelevant regions, then a token-level stage applies the original indexer only within the retained candidate blocks. The per-query cost drops from $\mathcal{O}(L)$ to $\mathcal{O}(L/B + mB)$. The optimization axis is the **sequence dimension**: fewer tokens are scored per query.

Indexer-head routing (MISA). Zhou et al. (2026) treat the H^I indexer heads as a pool of experts and route each query to a small subset of $h \ll H^I$ active heads via cheap block-level statistics. Only the selected heads execute token-level scoring, reducing the per-query cost from $\mathcal{O}(H^I{\cdot}L)$ to $\mathcal{O}(h \cdot L + H^I \cdot M)$ where $M = \lceil L/B \rceil$. The optimization axis is the **head dimension**: fewer heads are activated per query.

All three approaches reduce the per-invocation cost of the indexer while treating each invocation independently. None exploits the observation that adjacent decode steps produce highly overlapping top-k selections. In the following section, we present empirical evidence that this **temporal locality** is strong and structured, motivating a fourth, orthogonal optimization axis.

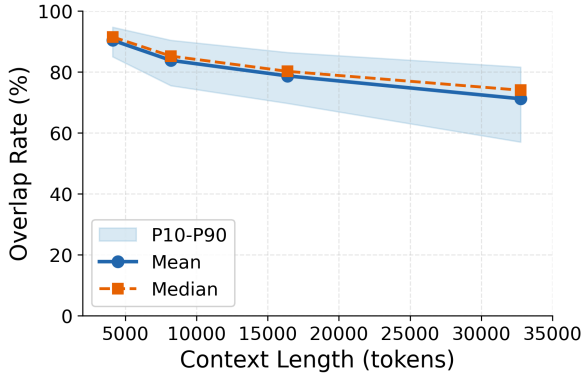

Figure 1: Average top-k overlap rate between adjacent decode steps as context length increases. The overlap decreases from 90.4% at 4K to 71.2% at 32K, but the degradation rate slows at longer contexts.

3 Temporal Locality in Sparse Indexing

We empirically characterize the temporal behavior of DSA's top-k selections across consecutive decode steps. All measurements are conducted on DeepSeek-V3.2 (671B parameters, FP8 inference, 8-way tensor parallelism on 8×H20 GPUs) with context lengths from 4K to 32K tokens and 128 decode steps.

3.1 High Overlap with Context-Length Degradation

We define the *overlap rate* between adjacent steps as $\frac{|\mathcal{T}_t \cap \mathcal{T}_{t+1}|}{k}$, measuring the fraction of top-k indices that persist from one step to the next. Figure 1 plots the average overlap rate across all layers and decode steps as a function of context length.

The overlap rate is substantial across all tested lengths, starting at 90.4% for 4K context and remaining above 71% even at 32K. However, a clear degradation trend emerges: as the context grows, the selection ratio k/L shrinks and boundary-score tokens become more interchangeable between steps, leading to lower raw overlap. The degradation follows an approximately logarithmic curve, suggesting it will not collapse at even longer contexts but nonetheless raises the question: does this 71% overlap accurately reflect the impact on attention quality?

3.2 Core Tokens Are Far More Stable

The plain overlap metric treats all top-k tokens equally, yet their contributions to the final attention output are highly unequal. To assess whether the observed changes affect high-contribution tokens, we measure the *top-N core stability*: the overlap rate restricted to the N highest-scoring tokens within each step's top-k set. Table 1 reports the results.

Table 1: Overlap rate between adjacent decode steps. *Plain*: unweighted set overlap over all k=2048 tokens. *Top-N*: overlap restricted to the N highest-scoring tokens.

Metric	4K	8K	16K	32K
Plain overlap	90.4	83.8	78.7	71.2
Top-512 stability	99.4	98.1	96.6	92.1
Top-256 stability	99.6	98.7	97.6	94.2
Top-128 stability	99.6	98.9	98.0	95.1

Figure 2: Per-bucket retention rate across context lengths. Each bucket contains 256 tokens ranked by descending index score (bucket 1 = highest). High-score tokens are nearly always retained; low-score boundary tokens account for the majority of changes.

The gap between plain overlap and core stability is striking. At 32K context, while 29% of the overall top-k set changes between steps, the top-128 highest-scoring tokens maintain 95.1% stability. This gap widens with context length (from 9% at 4K to 24% at 32K), indicating that longer contexts produce a sharper separation between a stable high-score core and a volatile low-score boundary.

Figure 2 further illustrates this by partitioning the 2048 selected tokens into 8 equal buckets by descending index score (bucket 1 = rank 1–256, highest scores; bucket 8 = rank 1793–2048, lowest scores) and measuring per-bucket retention rate across all four context lengths.

The retention rate drops monotonically from 94% (bucket 1) to 44% (bucket 8) at 32K, confirming that temporal instability is almost entirely a boundary phenomenon. This has a direct implica-

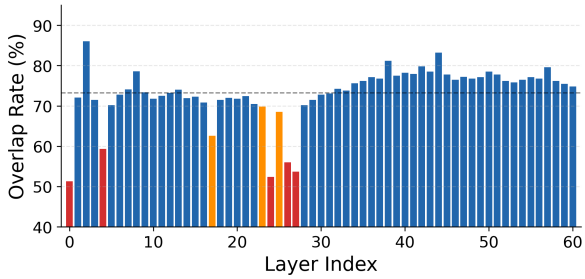

Figure 3: Per-layer plain overlap rate at 32K context. Red: layers below 60%; orange: 60–70%; blue: above 70%. The dashed line indicates the global mean.

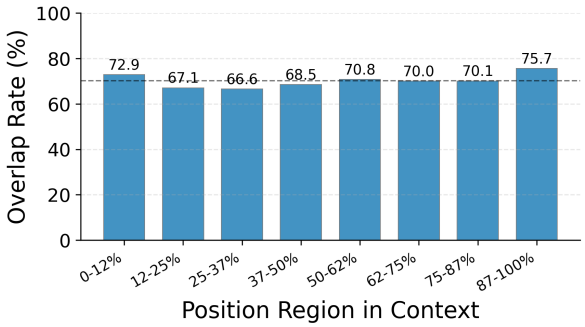

Figure 4: Per-region overlap rate at 32K context. The prefix is divided into 8 equal regions. The dashed line indicates the global mean (70.2%). All regions fall within a narrow band, confirming positional uniformity.

tion: even if partially stale cached scores cause a few boundary tokens to be incorrectly included or excluded, the high-contribution core tokens remain correctly selected, limiting the impact on attention quality.

3.3 Layer-Wise and Positional Patterns

Layer-wise heterogeneity. Figure 3 shows the per-layer plain overlap at 32K context. A cluster of layers (e.g., Layers 0, 24–27) exhibits overlap as low as 51%, while others (e.g., Layers 2, 42, 44) reach 86%. This heterogeneity is consistent across context lengths and suggests that layer-adaptive refresh strategies could further improve efficiency.

Positional uniformity. We further examine whether temporal instability concentrates at specific positions within the context. Figure 4 partitions the 32K prefix into 8 equal positional regions and reports per-region overlap rate. All regions fall within a narrow range of 67–76%, closely tracking the global mean of 71.2%, with no region exhibiting significantly higher or lower instability. This confirms that temporal changes are distributed uniformly across the prefix, validating the use of a simple round-robin partition strategy that treats all positions equally.

3.4 Summary

The analysis reveals three properties that jointly motivate temporal index reuse: **(1)** adjacent decode steps share the vast majority of high-value selections (95%+ for core tokens); **(2)** changes are confined to low-score boundary tokens with minimal attention contribution; **(3)** the temporal pattern is spatially uniform across positions, enabling simple round-robin partitioning. These findings suggest that refreshing only a fraction of index scores per step can preserve selection quality while substantially reducing computation.

4 Method: TISA

Motivated by the temporal locality established in Section 3, we propose TISA, which reduces the per-step indexer cost by maintaining a *score cache* and refreshing only a fraction of it at each decode step. The key idea is simple: partition all KV cache positions into K equal groups and recompute scores for exactly one group per step in a round-robin fashion, while reusing cached scores for the remaining $K-1$ groups. Figure 5 illustrates the contrast between the original DSA indexer and TISA.

4.1 Score Cache

For each layer, TISA maintains a persistent score cache $\hat{I} \in \mathbb{R}^{L_{\max}}$ that stores the most recently computed index score for every prefix position. At any decode step t with current prefix length L_t, the cache satisfies:

$$\hat{I}_s^{(t)} = I_{t',s}, \quad t' = \max\{\tau \le t \mid s \in \mathcal{P}_\tau\}, \tag{4}$$

where $\mathcal{P}_\tau$ denotes the set of positions whose scores were recomputed at step τ. The top-k selection then operates on the cached scores rather than freshly computed ones:

$$\hat{\mathcal{T}}_t = \mathrm{TopK}\left(\{\hat{I}_s^{(t)}\}_{s=1}^{L_t},\ k\right). \tag{5}$$

4.2 Partition and Refresh Schedule

Initialization (step 0). At the first decode step, TISA computes the full index score over all L_0 prefix positions and populates the cache:

$$\hat{I}_s^{(0)} = I_{0,s}, \quad \forall\, s \in \{1, \ldots, L_0\}. \tag{6}$$

This ensures the first generated token uses exact indexing with no approximation.

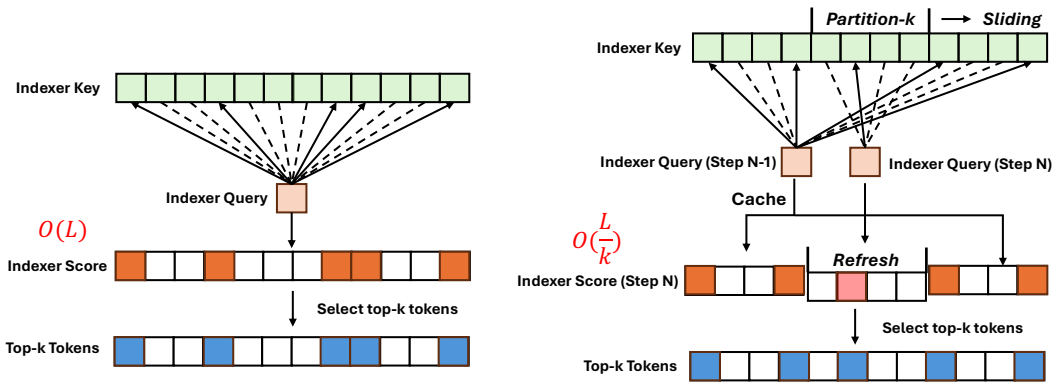

Figure 5: Comparison of the original DSA indexer (left) and TISA (right). DSA recomputes scores for all prefix positions at every decode step. TISA partitions positions into K groups and refreshes only one group per step, reusing cached scores for the rest.

Incremental update (step $t > 0$). The prefix positions $\{1, \ldots, L_t\}$ are evenly partitioned into K contiguous groups. Let the partition size be $P = \lceil L_t/K \rceil$. The partition assigned to step t is determined by round-robin:

$$\mathrm{id}(t) = (t-1) \bmod K, \tag{7}$$

and the corresponding position range is:

$$\mathcal{P}_t = \big[\mathrm{id}(t) \cdot P,\ \min\big(\mathrm{id}(t) \cdot P + P,\ L_t\big)\big)\,. \tag{8}$$

Only positions in $\mathcal{P}_t$ are rescored using the current query:

$$\hat{I}_s^{(t)} \leftarrow I_{t,s}, \quad \forall\, s \in \mathcal{P}_t. \tag{9}$$

Scores for positions outside $\mathcal{P}_t$ remain unchanged from the previous step.

New-position refresh. Each decode step appends a new token at position L_t to the KV cache. If this position falls outside the current partition $\mathcal{P}_t$, it would not be scored until its partition is scheduled, potentially missing recently generated tokens. TISA therefore unconditionally refreshes the newest position:

$$\hat{I}_{L_t}^{(t)} \leftarrow I_{t,L_t}. \tag{10}$$

This guarantees that every newly generated token is immediately visible to subsequent top-k selection. Algorithm 1 summarizes the complete procedure for a single layer during decode.

Algorithm 1 TISA: Temporal Index Reuse

Require: Score cache $\hat{I}$, partition count K, token budget k, decode step counter t, prefix length L_t
Ensure: Selected token set $\hat{\mathcal{T}}_t$ of size k

```
if t = 0 then
    Î_s ← I_{0,s} for all s ∈ {1, ..., L_0}      // Full computation
else
    P ← ⌈L_t/K⌉
    id ← (t − 1) mod K
    p_start ← id·P;   p_end ← min(p_start + P, L_t)
    Î_s ← I_{t,s} for all s ∈ [p_start, p_end)   // Refresh partition
    if L_t ∉ [p_start, p_end) then
        Î_{L_t} ← I_{t,L_t}                      // Refresh new position
    end if
end if
T̂_t ← TopK({Î_s}_{s=1}^{L_t}, k)
t ← t + 1
return T̂_t
```

4.3 Complexity Analysis

In the original DSA indexer, each decode step scores all L prefix positions, incurring $\mathcal{O}(L)$ per query per layer. TISA replaces this with scoring L/K positions (the active partition) plus one additional position (the new token), yielding a per-step cost of:

$$\mathcal{O}\left(\frac{L}{K} + 1\right) \approx \mathcal{O}\left(\frac{L}{K}\right). \tag{11}$$

Over K consecutive steps, every position is refreshed exactly once, so the amortized cost equals the original. The key benefit is that each *individual step* incurs only $1/K$ of the original indexer

computation. For decode-bound workloads where per-step latency directly determines token generation speed, this translates to up to $K\times$ reduction in indexer latency per step.

4.4 Composability with Spatial Optimizations

Since TISA only changes *how often* positions are scored without modifying the scoring mechanism itself, it composes naturally with all three spatial optimizations in Section 2.2. IndexCache reduces the number of layers that invoke the indexer; TISA further reduces each invocation's cost, yielding a multiplicative saving of $R\times K$ where R is the layer retention ratio. HISA narrows the search space to mB candidate tokens per query; TISA restricts this search to the active partition, shrinking the block-filtering stage from L/B to $L/(KB)$ blocks. MISA reduces the head count from H^I to h; TISA independently reduces the position count from L to L/K, as the two act on orthogonal dimensions of the same FP8 GEMM. In all cases, the only practical constraint is that partition boundaries align with block sizes, which is trivially satisfiable.

5 Experiments

We verify that TISA preserves downstream quality while delivering the expected kernel-level speedup.

5.1 Setup

Models. We conduct experiments on DeepSeek-V3.2 (DeepSeek-AI, 2025) and GLM-5 (GLM-5-Team, 2026), both of which adopt DSA-style token-level sparse attention with a lightweight indexer. All evaluations use the official FP8-quantized model checkpoints.

Configurations. The baseline is the unmodified DSA indexer. We test TISA at $K \in \{2, 4, 8\}$, corresponding to refreshing $1/2$, $1/4$, and $1/8$ of the score cache per step. The downstream Sparse MLA and its token budget ($k = 2048$) remain unchanged across all configurations.

Hardware. Quality evaluations run on 8× NVIDIA H20 GPUs (96GB) with 8-way tensor parallelism. Kernel benchmarks use a single H20 GPU with batch size 64, decode length $m = 1$, $H_I = 64$, $d_I = 128$.

Benchmarks. We report (i) generation quality on **LongBench** (Bai et al., 2023) across six task categories, (ii) retrieval accuracy on **Needle-in-a-Haystack** (NIAH) over contexts up to 128K tokens, and (iii) indexer kernel latency of our TileLang implementation (Wang et al., 2025).

5.2 LongBench

Table 2 summarizes LongBench performance. Since TISA only affects how frequently index scores are refreshed, the token budget k and the Sparse MLA operator are identical across all rows.

The two models respond differently to temporal reuse. On GLM-5, all K values match or slightly exceed the baseline average (55.48 $\rightarrow$ 55.72 at $K = 4$), indicating that its indexer produces temporally stable scores that tolerate aggressive caching. On DeepSeek-V3.2, the overall average decreases by 0.35 points at $K = 2$, 0.93 at $K = 4$, and 1.54 at $K = 8$. The degradation is gradual: doubling K from 2 to 4 costs 0.58 additional points, and doubling again from 4 to 8 costs 0.61 points. This sub-linear pattern is consistent with the core-token stability in Section 3: even at $K = 8$, high-scoring positions are refreshed frequently enough to maintain their rank, and only boundary tokens accumulate meaningful staleness.

Across task categories, QA, summarization, few-shot, and synthetic retrieval remain within noise-level variation on both models. The code completion category on DeepSeek-V3.2 is the notable exception, dropping from 42.99 to 37.08 as K grows from 1 to 8. Code generation involves rapidly shifting attention as the model navigates syntax structures and variable references, reducing temporal locality in the score distribution. However, this degradation does not appear on GLM-5 (72.24 $\rightarrow$ 72.58 at $K = 8$), suggesting it is model-specific rather than inherent to code tasks.

5.3 Needle-in-a-Haystack

We insert a target fact at controlled positions within a long distractor document and test whether the model can accurately recall it during generation. Context lengths range from 8K to 128K; insertion depths span 10% through 100% in 10% steps, yielding 50 (context, depth) pairs per configuration.

Figure 6 shows the accuracy heatmaps on DeepSeek-V3.2. Across all 200 tests (4 configurations $\times$ 50 pairs), only 2 cells show degradation (0.93): the baseline at (64K, depth = 100%) and K=4 at (32K, depth = 100%). Both occur at depth = 100%, where the needle is placed immediately before the question. K=2 and K=8 achieve perfect accuracy across all cells, matching or exceeding the baseline. The isolated K=4 degrada-

Table 2: LongBench results on DeepSeek-V3.2 and GLM-5. Scores are sub-task averages within each category. Abbreviations: **SQA** = Single-Doc QA, **MQA** = Multi-Doc QA, **Sum** = Summarization, **FS** = Few-shot, **Syn** = Synthetic, **Code** = Code Completion.

Model	Indexer	K	SQA	MQA	Sum	FS	Syn	Code	Avg.
DeepSeek-V3.2	DSA	–	**53.06**	**52.77**	22.37	**48.44**	62.67	**42.99**	**46.69**
	TISA	2	53.31	52.95	22.04	47.58	**64.00**	38.78	46.34
	TISA	4	52.09	52.14	**22.45**	47.95	62.67	37.20	45.76
	TISA	8	52.29	51.57	22.29	47.37	60.00	37.08	45.15
GLM-5	DSA	–	52.41	52.21	24.29	**70.25**	74.67	72.24	55.48
	TISA	2	52.39	52.47	24.51	69.10	77.33	72.00	55.71
	TISA	4	**52.51**	52.19	**24.60**	68.28	**77.33**	**73.90**	**55.72**
	TISA	8	51.67	**53.88**	24.17	68.57	76.00	72.58	55.54

tion at a different context length than the baseline indicates random variation rather than systematic harm from stale scores. This confirms that high-relevance tokens remain stably within the top-k selection even when their partition is not actively refreshed, consistent with the core-token stability observed in Section 3.

5.4 Kernel-Level Speedup

TISA does not require a custom GPU kernel: it invokes the same TileLang FP8 indexer kernel as DSA but passes only the active partition slice (L/K positions) as input each step. The kernel grid shrinks proportionally, and scored results are written back into a persistent float32 score cache. We time the indexer scoring pass alone (excluding top-k selection) using CUDA events over 1000 iterations. Figure 7 reports wall-clock latency.

The measured speedup approaches the theoretical $K\times$ ceiling as context length grows. At 128K, $K = 2$ achieves $1.91\times$, $K = 4$ achieves $3.81\times$, and $K = 8$ achieves $7.28\times$. The remaining gap stems from fixed per-block overhead: each thread block loads the query matrix (8KB) from global memory regardless of how many positions it scores, and the two-stage pipeline incurs startup/drain cost per block. At shorter contexts where fewer thread blocks are launched, GPU occupancy is lower and the gap widens (e.g., $K = 8$ at 8K yields $4.9\times$ rather than $8\times$).

Deployment projection. DeepSeek-V3.2 runs the indexer independently at all 61 layers per decode step. At 128K context with $K = 4$, the total indexer cost drops from $581 \times 61 \approx 35.5$ ms to $153 \times 61 \approx 9.3$ ms per token, saving over 26 ms per generated token. Combined with the quality results above, $K = 4$ provides a favorable operating point: $3.8\times$ indexer speedup with LongBench scores within 1 point of the baseline.

6 Related Work

Sparse attention mechanisms. Sparse attention reduces the quadratic cost of self-attention by restricting each query to attend over a subset of tokens. Early work established fixed sparse patterns: Longformer (Beltagy et al., 2020) augments sliding-window attention with task-specific global tokens, and Reformer (Kitaev et al., 2020) uses locality-sensitive hashing to approximate full attention. These architectures require training with the sparse pattern, limiting their applicability to pretrained dense models. More recent methods target inference-time sparsity without retraining. MoBA (Lu et al., 2025) applies mixture-of-experts-style routing over blocks, NSA (Yuan et al., 2025) combines compression, selection, and sliding-window branches, and MInference (Jiang et al., 2024) profiles per-head sparse patterns offline. These block-level methods are hardware-friendly but sacrifice selection granularity. Token-level methods offer finer control: DSA (DeepSeek-AI, 2025) introduces a lightweight indexer that scores every prefix token and selects the top-k for downstream attention, achieving strictly finer-grained sparsity than block-level approaches. Our work does not propose a new sparse attention pattern; rather, we accelerate the indexer that produces the pattern.

KV cache optimization. An orthogonal line of work reduces attention cost by compressing or evicting entries from the KV cache itself. H_2O (Zhang et al., 2023) maintains a fixed-size

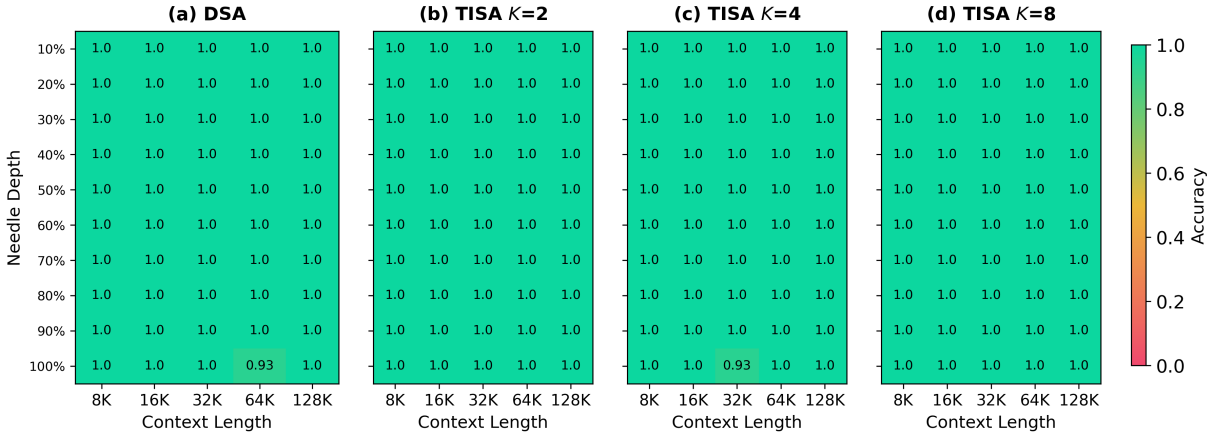

Figure 6: Needle-in-a-Haystack accuracy on DeepSeek-V3.2. Each cell reports retrieval accuracy at a given (context length, needle depth) pair. All TISA configurations match the baseline, indicating that temporal score reuse does not impair retrieval.

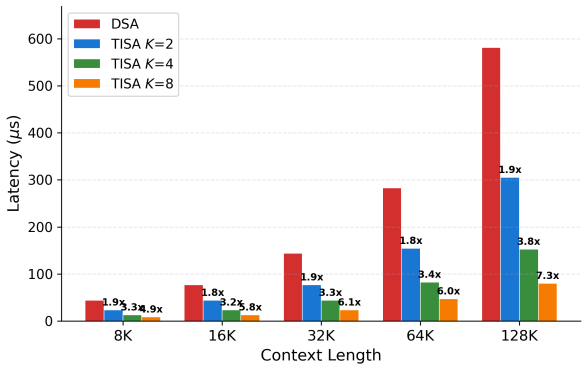

Figure 7: Indexer kernel latency on a single NVIDIA H20 GPU across context lengths. Numbers above each bar indicate speedup relative to DSA.

cache by evicting tokens with the lowest cumulative attention score. StreamingLLM (Xiao et al., 2024) retains only attention sinks and a sliding window, enabling infinite-length streaming at the cost of discarding middle context. SnapKV (Li et al., 2024) uses an observation window to identify important positions before generation and compresses the cache accordingly. Quest (Tang et al., 2024) estimates per-chunk importance to skip irrelevant KV pages during decode. These methods irreversibly discard or compress information, trading recall for efficiency.

Indexer acceleration. The most closely related work targets the indexer bottleneck of token-level sparse attention. IndexCache (Bai et al., 2026) exploits cross-layer similarity by sharing indices from a few full layers to neighboring shared layers, reducing the number of indexer invocations along the *layer* axis. HISA (Xu et al., 2026) introduces a hierarchical coarse-to-fine search that prunes irrelevant blocks before token-level scoring, reducing the per-query search space along the *sequence* axis. MISA (Zhou et al., 2026) routes each query to a subset of indexer heads, reducing computation along the *head* axis. All three optimize within a single indexer invocation, treating consecutive decode steps independently. TISA identifies and exploits a fourth axis: *temporal* redundancy across decode steps. By caching and incrementally refreshing index scores, TISA reduces per-step computation without modifying the indexer's internal structure, making it complementary to all three spatial approaches.

7 Conclusion

We presented TISA, which exploits temporal locality in index scores across consecutive decode steps to reduce the per-step cost of the DSA indexer. By partitioning prefix positions into K groups and refreshing one group per step in round-robin fashion, TISA lowers indexer computation by up to $K\times$ while maintaining a complete score cache for top-k selection. Experiments confirm that this strategy preserves quality on LongBench and NIAH, with kernel speed up to $7.3\times$. Notably, TISA requires no retraining, no architectural modification, and no custom GPU kernel. It operates along the temporal axis, orthogonal to existing spatial optimizations (IndexCache, HISA, MISA), and can be composed with them without conflict. As context windows scale toward 1M tokens, TISA demonstrates that the temporal dimension offers a new and effective avenue for improving efficiency of sparse attention.

Limitations

TISA applies exclusively to the decode phase because no temporal history exists to reuse during prefill. The uniform round-robin partition does not account for per-layer variation in temporal stability: layers with lower overlap may benefit from more frequent refresh. Additionally, an adaptive schedule that assigns smaller K to volatile layers could improve the quality-speed trade-off but introduces additional complexity. Finally, our quality evaluation is limited to contexts up to 128K; behavior at longer contexts (e.g., 1M tokens) where temporal stability may further degrade remains to be verified.

References

Yushi Bai, Qian Dong, Ting Jiang, Xin Lv, Zhengxiao Du, Aohan Zeng, Jie Tang, and Juanzi Li. 2026. Indexcache: Accelerating sparse attention via cross-layer index reuse. *arXiv preprint arXiv:2603.12201*.

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. 2023. LongBench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

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

DeepSeek-AI. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

DeepSeek-AI. 2026. Deepseek-v4: Towards highly efficient million-token context intelligence.

GLM-5-Team. 2026. Glm-5: from vibe coding to agentic engineering. *Preprint*, arXiv:2602.15763.

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. 2024. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. *arXiv preprint arXiv:2407.02490*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, and 6 others. 2025. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*.

Qwen. 2025. Qwen2.5-1m: Deploy your own qwen with context length up to 1m tokens.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning*.

Lei Wang, Yu Cheng, Yining Shi, Zhengju Tang, Zhiwen Mo, Wenhao Xie, Lingxiao Ma, Yuqing Xia, Jilong Xue, Fan Yang, and Zhi Yang. 2025. Tilelang: A composable tile-based programming model for ai systems. *arXiv preprint arXiv:2504.17577*.

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

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. 2026. Hisa: Efficient hierarchical indexing for fine-grained sparse attention. *arXiv preprint arXiv:2603.28458*.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, and 1 others. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*.

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

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

A The Use of Large Language Models

In this work, large language models are used exclusively to assist with language editing and clarification during the writing of this paper. All technical ideas, method design, analysis, and experimental work are conducted by human authors.