LISA: HEAD-FUSED LINEAR INDEXING FOR EFFICIENT SPARSE ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse attention reduces the cost of long-context inference by restricting attention to a small subset of tokens for each query. However, identifying this subset can still be expensive: the indexer in DeepSeek Sparse Attention (DSA) scores every visible token with multiple query heads, leaving substantial indexing overhead during long-context prefilling. We introduce LISA, a training-free indexer based on an exact decomposition of the weighted-ReLU score into a linear term and a nonlinear correction. Shared keys allow the linear contributions of all heads to be fused exactly into one dot product per token. LISA directly selects tokens using this linear term, omitting the nonlinear correction and removing the head-count factor from full-context scoring while preserving token-level granularity. LISA† adds candidate refinement with the original multi-head scorer, while LISA‡ shares candidate pools across layers with layer-specific refinement. Across 4K–128K contexts, LISA maintains RULER scores close to the respective native baselines on DeepSeek-V3.2 and GLM-5.2. On DeepSeek-V3.2 with a 1M-token input, LISA achieves a $4.13\times$ speedup over DSA in single-request offline time to first token, while LISA‡ with four-layer groups achieves $5.20\times$. These results support head-fused linear scoring as an efficient approximation to multi-head indexing for long-context prefilling.

1 INTRODUCTION

Long-context processing is important for large language models (LLMs) in applications such as document question answering and software engineering over large codebases. Answering questions about long documents can require integrating evidence from distant passages (Bai et al., 2024), while resolving software issues can require reasoning across multiple source files (Jimenez et al., 2024). During prefill, dense self-attention computes interactions between each query and all causally visible keys, so its computational cost grows quadratically with sequence length. IO-aware implementations such as FlashAttention reduce memory traffic and accelerate exact attention, but retain its quadratic computational complexity (Dao et al., 2022). As inputs grow longer, this attention computation increases the amount of work required before generation begins and leads to a longer time to first token (TTFT), motivating methods that reduce the cost of long-context prefilling.

Sparse attention reduces attention computation by restricting each query to a small subset of key-value entries (Child et al., 2019). In token-level sparse attention, this subset can include individual positions throughout the visible context, allowing each query to retain distant tokens without attending to all intervening positions. DeepSeek Sparse Attention (DSA), used in DeepSeek-V3.2, employs a learned lightning indexer to select the top-k tokens for each query (DeepSeek-AI, 2025). Sparse Multi-Head Latent Attention (Sparse MLA) then computes attention only over this selected subset, reducing the main attention computation during prefill from $\mathcal{O}(L^2)$ to $\mathcal{O}(Lk)$ for sequence length L and $k \ll L$. However, selecting this subset still requires the indexer to score the visible context across multiple heads, leaving a full-context token-scoring stage outside the sparse attention computation.

Existing approaches reduce this indexing overhead along different dimensions. IndexCache (Bai et al., 2026) reduces repeated indexing through cross-layer reuse and LiteTopK (Yin et al., 2026) reduces selection overhead. HISA (Xu et al., 2026) reduces the token search space through hierarchical block screening, and MISA (Zhou et al., 2026) reduces the number of indexer heads used for each query. These approaches reduce how often, where, or across how many tokens and heads the original indexer

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.

is evaluated. Despite these optimizations, whenever token-level scores are computed, the underlying operation remains head-wise ReLU-activated query-key dot products.

We introduce LISA to directly accelerate this token-level scoring. Our key insight is the identity: $\mathrm{ReLU}(x) = x + \mathrm{ReLU}(-x)$. Applying this identity to the original weighted-ReLU score yields an exact decomposition into a linear term and a head-wise nonlinear correction. We compute the linear term by first summing the query vectors across heads with their original weights, then taking one dot product between this fused query and each token's key. This removes the head-count factor from full-context linear scoring while preserving an individual score for each token. LISA directly selects the final top-k tokens using this linear score, omitting the nonlinear correction. Empirically, these combined linear contributions retain useful information for token selection. LISA reuses the pretrained indexer parameters and preserves the selection budget and token-index interface to Sparse MLA, requiring no additional training.

To further improve selection quality, we introduce LISA†, which first retains a broader candidate set using the head-fused linear score and then selects the final tokens with the original multi-head scorer. This refinement incorporates the omitted nonlinear corrections into the final selection while restricting full multi-head scoring to the candidates. To reduce repeated screening, LISA‡ shares candidate pools across adjacent layers while each layer selects tokens with its own scorer.

Our contributions are:

- We introduce LISA, a training-free indexer that fuses the weighted linear contributions of all indexer heads, removing the head-count factor from full-context scoring while preserving token-level selection. We extend this design with candidate refinement in LISA† and cross-layer candidate-pool sharing in LISA‡.
- We implement LISA and its variants. With 64 indexer heads and 1M keys, LISA achieves a $6.12\times$ speedup over DSA for the complete selector. On DeepSeek-V3.2 with a 1M-token input, LISA achieves a $4.13\times$ speedup over DSA in single-request offline TTFT. With candidate-pool sharing, LISA‡ achieves a $5.20\times$ TTFT speedup over DSA under the same conditions.
- Across 4K–128K contexts, LISA maintains RULER (Hsieh et al., 2024) scores close to the respective native baselines on DeepSeek-V3.2 and GLM-5.2. Single-head comparisons show that head fusion retains more attention mass than either fixed or query-dependent single-head selection, and recall analyses quantify the effects of candidate refinement and cross-layer candidate-pool sharing.

2 Related Work

Sparse attention. Sparse attention reduces the computational cost of long-context attention by restricting each query to a subset of key–value entries (Child et al., 2019). Native Sparse Attention (NSA) (Yuan et al., 2025) combines separate compression, selection, and sliding-window branches, whereas MoBA (Lu et al., 2025) routes each query to KV blocks based on its inner products with mean-pooled block keys. MiniMax Sparse Attention (MSA) (Lai et al., 2026) uses a lightweight indexing branch to select KV blocks independently for each grouped-query attention (GQA) group, with the main branch attending to the tokens within the selected blocks. DSA (DeepSeek-AI, 2025) uses a learned lightning indexer to score individual tokens and select the top-k key–value entries for Sparse MLA. LISA builds on this token-level selection interface and reduces the computational cost of index scoring through head fusion.

Efficient sparse indexing. Token selection in DSA still requires full-context scoring and top-k selection, motivating optimizations of both operations (DeepSeek-AI, 2025). LiteTopK (Yin et al., 2026) retains the original multi-head scoring function and exact top-k selection under that function, while fusing threshold filtering into the scoring pipeline to reduce intermediate score writes and selection overhead. HISA (Xu et al., 2026) first filters blocks using mean-pooled indexing keys, then refines tokens within the selected blocks with the original multi-head scorer. PIVOT (Liu et al., 2026) aggregates neighboring queries into a shared proxy to reduce repeated prefix scans; its Refine variant then applies the original scorer separately for each query within a shared candidate set. MISA (Zhou et al., 2026) instead uses block-level statistics to route each query to a small subset of

the original indexer heads, which then perform token-level scoring. Its hierarchical variant, MISA†, uses this reduced-head scorer to form an enlarged token candidate set and then refines the candidates with all original indexer heads. LISA instead exploits shared keys to fuse the weighted query vectors across all heads at each query position and directly selects the final tokens using the resulting linear score. LISA† extends this design by first retaining a broader candidate set with the fused linear score and then selecting the final tokens with the original multi-head scorer.

Cross-layer index reuse. Cross-layer sharing reduces repeated indexing work by reusing final token selections or candidate pools. IndexCache (Bai et al., 2026) partitions layers into Full and Shared layers: Full layers compute their own top-k indices, while Shared layers reuse those from the nearest preceding Full layer. Cross-Layer Sparse Attention (CLSA) (Sun et al., 2026) trains a shared indexer within a KV-sharing cross-decoder architecture, computing token selections once per query for reuse across cross-decoder layers. LongCat Sparse Attention (LSA) (Zan et al., 2026) combines cross-layer index reuse with coarse page-level filtering and fine token-level scoring, reducing both repeated indexer evaluations and the number of tokens requiring fine-grained scoring. DeepSeek-V4.1-Flash (DeepSeek-AI, 2026) supports both index reuse and reindexing in CSA2: some layers reuse existing top-k indices, while its decoder's hierarchical indexer lets subsequent reindexing layers rescore a shared candidate pool and produce their own selections. LISA‡ constructs shared candidate pools using head-fused linear scores, while each layer independently selects its final tokens with its own original multi-head scorer.

3 METHOD

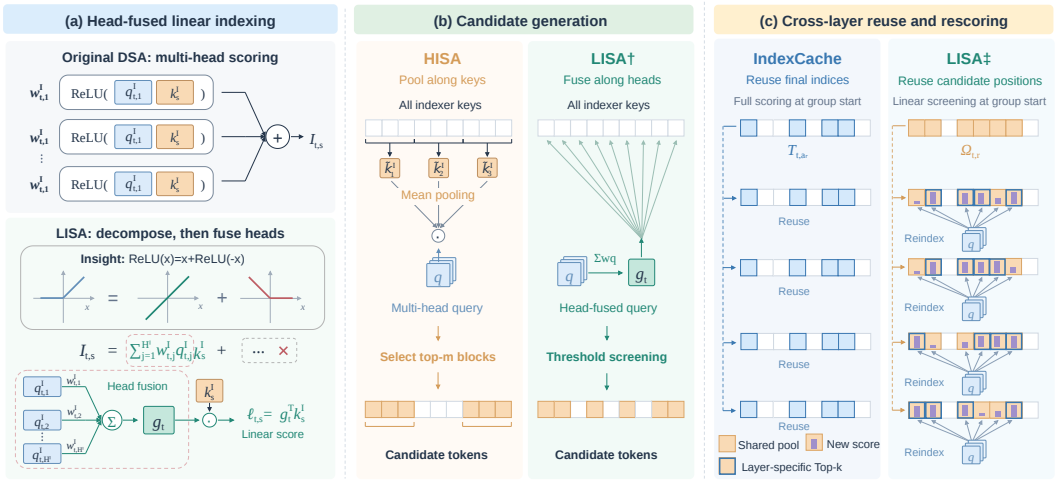

Figure 1: Overview of LISA and its variants. (a) LISA fuses weighted queries across heads for linear scoring and direct top-k token selection. (b) LISA† screens tokens before refinement with the original multi-head scorer, whereas HISA screens blocks using mean-pooled keys. (c) LISA‡ shares candidate positions for layer-specific rescoring and selection, whereas IndexCache reuses final indices.

LISA exploits the indexer's shared-key structure through head fusion, combining query vectors across all heads with their original weights before scanning the context. This produces a linear score with one dot product per token, removing the head-count factor from the full-context scan while preserving token-level scoring. The basic LISA variant directly selects the final tokens using this score; to improve selection quality, the hierarchical variant LISA† first retains a broader candidate set and refines it with the original multi-head scorer. LISA‡ further reduces repeated full-context scans by sharing candidate pools across adjacent layers, while each layer independently scores the candidates and selects its final tokens. Figure 1 summarizes these designs.

3.1 LISA: HEAD-FUSED LINEAR INDEXING

Original indexer. For a query at position t, let $\mathcal{V}_t$ denote its causally visible key positions, with $N_t = |\mathcal{V}_t|$. The original DSA indexer (DeepSeek-AI, 2025) assigns each visible key the score

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

Here, H^I is the number of indexer heads, $\mathbf{q}_{t,j}^I \in \mathbb{R}^{D^I}$ are the query vectors, and $w_{t,j}^I \in \mathbb{R}$ are query-dependent, possibly signed weights. All heads share the same key $\mathbf{k}_s^I \in \mathbb{R}^{D^I}$ at each visible position s. Scoring all N_t visible keys costs $\mathcal{O}(N_t H^I D^I)$ per query. The indexer then selects the highest-scoring $\min(k, N_t)$ positions, where k is the selection budget. Although the key is shared, ReLU acts on each head's dot product before the weighted sum. Consequently, aggregating the query vectors directly does not in general reproduce the original score.

Insight: Head Fusion. Using the identity $\mathrm{ReLU}(x) = x + \mathrm{ReLU}(-x)$, we separate the original score $I_{t,s}$ into a linear term and a nonlinear correction:

$$I_{t,s} = \ell_{t,s} + c_{t,s}, \tag{2}$$

$$\ell_{t,s} = \sum_{j=1}^{H^I} w_{t,j}^I (\mathbf{q}_{t,j}^I)^\top \mathbf{k}_s^I, \tag{3}$$

$$c_{t,s} = \sum_{j=1}^{H^I} w_{t,j}^I \,\mathrm{ReLU}\big(-(\mathbf{q}_{t,j}^I)^\top \mathbf{k}_s^I\big)\,. \tag{4}$$

Because all heads share the same key vector, we can move the weighted head aggregation before the dot product and define a fused query (Figure 1(a)):

$$\mathbf{g}_t = \sum_{j=1}^{H^I} w_{t,j}^I \mathbf{q}_{t,j}^I, \tag{5}$$

$$\ell_{t,s} = \mathbf{g}_t^\top \mathbf{k}_s^I, \qquad s \in \mathcal{V}_t. \tag{6}$$

With one fusion per query, LISA's scoring cost is $\mathcal{O}(H^I D^I + N_t D^I)$. Each token still receives an individual score, preserving the original indexer's token-level granularity.

Direct token selection. LISA directly selects the final $k_t = \min(k, N_t)$ token positions using the linear score:

$$\mathcal{T}_t^{\mathrm{LISA}} = \mathrm{TopK}(\{\ell_{t,s} \mid s \in \mathcal{V}_t\},\, k_t)\,. \tag{7}$$

Here, TopK returns token indices, which are passed to Sparse MLA for attention computation. LISA uses the linear term $\ell_{t,s}$ to approximate $I_{t,s}$, omitting the nonlinear correction $c_{t,s}$.

3.2 HIERARCHICAL LISA

To reduce the effect of omitting the nonlinear correction on final token selection, we extend LISA to a hierarchical variant, LISA†. Given a final selection budget k and a target candidate count $M \geq k$, LISA† first screens tokens by thresholding the same linear score (Figure 1(b)); the actual number of retained candidates can vary across queries. It then evaluates the original multi-head score only within this candidate set and selects the final $k_t = \min(k, N_t)$ tokens.

Candidate construction. For queries with $N_t > M$, we adapt the sampling-and-histogram approach of LiteTopK (Yin et al., 2026) to estimate an initial threshold $\tau_t^{(0)}$ from sampled linear scores, targeting approximately M candidates. We apply this threshold to all visible positions to form the initial candidate set:

$$\Omega_t^{(0)} = \Big\{ s \in \mathcal{V}_t \;\Big|\; \ell_{t,s} \geq \tau_t^{(0)} \Big\}\,. \tag{8}$$

When $N_t \le M$, all visible positions enter $\Omega_t^{(0)}$ directly. Let $C \ge M$ be the candidate-count limit. If the initial count falls outside $[k_t, \min(C, N_t)]$, we adjust the cutoff and, when necessary, break score ties deterministically to bring the count within these bounds. We denote the resulting candidate set by Ω_t, with $M_t = |\Omega_t|$; Appendices B.2 and B.3 detail threshold estimation and count recovery.

Candidate refinement. Given the candidate set Ω_t, LISA† evaluates the original score $I_{t,s}$ in Eq. (1) for each $s \in \Omega_t$, using all H^I query heads. This refinement brings the total scoring cost to $\mathcal{O}(H^I D^I + N_t D^I + M_t H^I D^I)$ per query. It then selects the $k_t = \min(k, N_t)$ highest-scoring positions within this set:

$$\mathcal{T}_t^{\mathrm{LISA}^\dagger} = \mathrm{TopK}(\{I_{t,s} \mid s \in \Omega_t\},\, k_t)\,. \tag{9}$$

The selected indices are passed to Sparse MLA through the same interface as in LISA. Algorithm 1 (Appendix B.1) summarizes the single-layer procedures for LISA and LISA†.

3.3 LISA WITH SHARED CANDIDATE POOLS

LISA† reduces the cost of full-context scoring through head fusion, but still constructs a candidate set independently at each layer. Motivated by the cross-layer similarity of token selections observed in prior work (Bai et al., 2026), we introduce LISA‡ to share candidate pools across adjacent layers. Each layer uses its own original multi-head scorer to independently select the final tokens from this shared candidate pool.

We partition the model layers into groups of consecutive layers $\mathcal{G}_r$, with a_r denoting the first layer of group r. For a query at position t, layer a_r constructs the shared candidate set $\Omega_{t,r}$ from its own head-fused linear scores, following the threshold screening and any necessary candidate-count adjustment in Section 3.2. The resulting pool has $M_{t,r} = |\Omega_{t,r}|$ candidates; it remains fixed within $\mathcal{G}_r$ and serves as the common candidate set for each layer's independent refinement.

For each layer $l \in \mathcal{G}_r$, we compute the original multi-head score $I_{t,s}^{(l)}$ in Eq. (1) for $s \in \Omega_{t,r}$ using that layer's own queries, keys, and head weights. Each layer then selects the k_t highest-scoring positions within this pool:

$$\mathcal{T}_{t,l}^{\mathrm{LISA}^\ddagger} = \mathrm{TopK}\Big(\{I_{t,s}^{(l)} \mid s \in \Omega_{t,r}\},\, k_t\Big)\,, \qquad l \in \mathcal{G}_r. \tag{10}$$

The resulting token sets can differ across layers (Figure 1(c)), and each layer passes its selected indices to its own Sparse MLA operator. Sharing the pool gives an average per-layer scoring cost of $\mathcal{O}((H^I D^I + N_t D^I)/|\mathcal{G}_r| + M_{t,r} H^I D^I)$ per query. Algorithm 2 (Appendix B.4) gives the cross-layer procedure for LISA‡.

4 EXPERIMENTS

We first evaluate LISA through complete indexer latency, end-to-end time to first token (TTFT), and long-context task performance on RULER (Sections 4.1–4.3). We then analyze attention-mass recall to examine the token-selection quality of head-fused linear indexing (Section 4.4). We use these results to discuss the cost and recall of LISA†, and compare candidate-pool sharing in LISA‡ with cross-layer reuse of final indices. Finally, score visualizations relate these observations to the linear and nonlinear terms of the original indexer score (Section 4.5).

Experimental setup. We evaluate LISA and LISA† on DeepSeek-V3.2 (DeepSeek-AI, 2025) and GLM-5.2 (GLM-5 Team, 2026)[1] using pretrained checkpoints without additional training. Their native indexers use $H^I = 64$ and $H^I = 32$ query heads, respectively, with $D^I = 128$ in both models. On GLM-5.2, all evaluated methods retain the official IndexCache schedule for reusing final token indices across layers. All compared methods use a final selection budget of $k = 2048$ tokens per query. For LISA† and LISA‡, we use a target candidate count of $M = 8192$ and a candidate-count limit of $C = 16384$. Both variants estimate screening thresholds using a sampling budget of 2048 token positions and a 256-bin histogram. We evaluate LISA‡ on DeepSeek-V3.2. IndexCache and LISA‡ use a group size of $G = 4$ in all main-text experiments.

[1] https://huggingface.co/zai-org/GLM-5.2

Compared methods. We use DSA as the native baseline on DeepSeek-V3.2 and IndexCache on GLM-5.2. For efficiency, we compare with the native baselines and LiteTopK (Yin et al., 2026), which retains full multi-head scoring while accelerating token selection. For quality evaluation, we include HISA (Xu et al., 2026) with 128-token blocks and a candidate budget of 8192 tokens. On DeepSeek-V3.2, we also compare with IndexCache (Bai et al., 2026) using the official uniform grouping. On DeepSeek-V3.2, IndexCache and LISA‡ treat the first layer independently and partition the remaining layers into consecutive groups of G; IndexCache reuses the final indices selected by each group's first layer. For recall analysis, we additionally include fixed single-head DSA and MISA (Zhou et al., 2026) with $h = 1$; MISA selects one head dynamically for each query. We calibrate the fixed head on 130 samples from the 32K RULER subset (10 per task), averaging $|w^I_{t,j}|\|\mathbf{q}^I_{t,j}\|_2$ over samples and layers that compute indices. For each model, we select the head with the largest mean and use that head ID for all subsequent indexer evaluations.

4.1 Indexer Kernel Performance

Table 1 compares the indexer latency of DSA, LiteTopK, LISA, and LISA†. Each invocation processes the last 8192 causal queries of a 16K–1M context using synthetic FP8 inputs, with 32 or 64 indexer heads. Our LISA implementation adapts the selection pipeline of LiteTopK (Yin et al., 2026) to the head-fused linear score.

Table 1: Median indexer latency (ms). Lower is better.

	$H^I = 32$				$H^I = 64$			
Length	DSA	LiteTopK	LISA	LISA†	DSA	LiteTopK	LISA	LISA†
16K	0.79	0.80	**0.64**	2.18	1.14	1.14	**0.65**	2.70
32K	1.63	1.63	**1.14**	2.33	2.28	2.29	**1.09**	2.83
64K	3.17	4.35	**2.34**	2.73	4.50	5.63	**2.24**	3.21
128K	6.15	7.77	**3.40**	3.51	9.19	10.44	**3.36**	3.99
256K	12.93	12.85	**4.94**	5.07	18.40	19.00	**5.49**	5.64
512K	26.26	22.04	**7.51**	8.17	40.75	35.03	**8.62**	8.88
1M	56.20	39.23	**13.78**	15.65	84.00	67.92	**13.73**	17.03

LISA is faster than both baselines at every tested context length from 16K to 1M for both head counts. With 64 heads, LISA achieves speedups of $6.12\times$ over DSA and $4.95\times$ over LiteTopK at 1M. With 32 heads, the corresponding speedups are $4.08\times$ and $2.85\times$.

LISA† has higher latency than LISA at every tested length for both head counts. It is slower than both baselines at 16K and 32K, but faster than both from 64K to 1M. This short-context slowdown is consistent with the added overhead of candidate management and multi-head refinement being harder to amortize. The fixed candidate cap bounds refinement work, while full-context scoring grows.

4.2 End-to-End Performance

We measure time to first token (TTFT) on DeepSeek-V3.2 and GLM-5.2 using vLLM (Kwon et al., 2023) with tensor and expert parallelism ($\mathrm{TP} = \mathrm{EP} = 8$), FP8 weights, and FP8 KV caches. We time the complete generation call for one pretokenized request with 16K–1M input tokens and one output token. For DeepSeek-V3.2 at 256K–1M, we use YaRN (Peng et al., 2024) with the same settings across methods at each length.

LISA reduces end-to-end TTFT on both models, with larger speedups at longer contexts (Table 2). At 1M on DeepSeek-V3.2, it reduces TTFT from 383.16 s to 92.72 s and achieves speedups of $4.13\times$ over DSA and $3.71\times$ over LiteTopK. With native IndexCache retained on GLM-5.2, the corresponding 1M speedups are $1.94\times$ over IndexCache and $1.50\times$ over LiteTopK.

LISA† has higher TTFT than LISA at every tested length on both models, but outperforms both baselines from 128K to 1M. TTFT includes early prefill chunks with short visible prefixes, whereas the kernel benchmark measures only the final 8192 queries. This helps explain why LISA† remains slower than both baselines at 64K despite lower indexer latency.

Table 2: Median offline TTFT (s). Lower is better.

	GLM-5.2				DeepSeek-V3.2				
Length	IndexCache	LiteTopK	LISA	LISA†	DSA	LiteTopK	LISA	LISA†	LISA‡
16K	**0.83**	**0.83**	**0.83**	0.88	0.82	0.81	**0.79**	1.00	0.91
32K	1.69	1.70	**1.67**	1.79	1.77	1.77	**1.64**	2.07	1.88
64K	3.51	3.51	**3.41**	3.61	4.08	4.10	**3.48**	4.28	3.82
128K	7.47	7.41	**7.00**	7.29	10.57	10.50	**7.49**	8.89	7.79
256K	17.04	16.23	**14.54**	15.07	31.16	30.02	16.83	19.20	**16.01**
512K	43.21	38.25	**30.80**	32.14	101.41	94.56	37.39	44.15	**33.63**
1M	131.15	101.22	**67.67**	72.96	383.16	344.12	92.72	113.49	**73.67**

By sharing candidate pools, LISA‡ reduces TTFT relative to LISA† at every tested length on DeepSeek-V3.2, with larger savings at longer contexts. It also outperforms LISA from 256K onward, suggesting that reduced screening cost offsets the extra refinement work. At 1M, its TTFT is 73.67 s versus 113.49 s for LISA†, giving a $5.20\times$ speedup over DSA.

4.3 Long-Context Evaluation on RULER

We evaluate long-context task performance on RULER (Hsieh et al., 2024), which covers retrieval, multi-hop tracing, aggregation, and question answering. We test both models at six context lengths from 4K to 128K, using 100 examples per task for all 13 tasks at each length. We apply the HELMET scoring rules (Yen et al., 2025) and average scores equally over tasks at each length.

Table 3: RULER scores (%) on DeepSeek-V3.2 and GLM-5.2 across six context lengths.

Model	Method	4K	8K	16K	32K	64K	128K	Avg.
DeepSeek-V3.2	DSA	97.17	96.95	97.44	**97.49**	95.92	94.34	96.55
	IndexCache	97.25	**97.02**	**97.58**	97.42	95.79	94.02	96.52
	HISA	97.18	96.82	97.24	91.38	73.51	60.87	86.17
	LISA	**97.30**	96.69	97.35	97.30	95.70	**94.82**	96.53
	LISA†	97.21	97.00	97.20	97.28	95.97	94.37	96.50
	LISA‡	97.16	96.71	97.35	97.40	**96.44**	94.41	**96.58**
GLM-5.2	IndexCache	97.91	97.80	97.67	**97.83**	97.31	96.87	97.56
	HISA	**98.14**	**98.18**	96.97	91.55	82.62	75.21	90.45
	LISA	97.78	97.88	97.45	97.69	97.00	96.72	97.42
	LISA†	97.91	98.03	**97.68**	97.69	**97.62**	**97.08**	**97.67**

LISA achieves RULER scores comparable to the respective native baselines on both models (Table 3). Its average score is 96.53 versus 96.55 for DSA on DeepSeek-V3.2, and 97.42 versus 97.56 for IndexCache on GLM-5.2. At each tested context length, the absolute score difference from the respective native baseline is below 0.5 percentage points on both models. LISA's DeepSeek-V3.2 average is also close to IndexCache's 96.52. LISA uses token-level selection rather than HISA's block-level candidate screening, and achieves higher RULER scores from 32K onward on both models. Together with the speedups reported above, these results show that head-fused linear indexing can reduce indexing overhead while retaining strong performance on the evaluated long-context tasks.

LISA† obtains average scores of 96.50 on DeepSeek-V3.2 and 97.67 on GLM-5.2, close to those of LISA and the respective native baselines. LISA‡ also retains comparable task performance on DeepSeek-V3.2, averaging 96.58 while sharing candidate pools across layers. We next examine whether these similar RULER scores correspond to similar attention-mass retention (Section 4.4).

4.4 Selection Quality Analysis

We measure attention-mass recall at the first token of the question body in each RULER example. At a fixed layer and query position t, let $p_{h,i}$ denote the dense-attention probability assigned by

core-attention head h to token i. For a method's selected token set $\mathcal{S}$, we compute

$$R(\mathcal{S}) = \frac{1}{H}\sum_{h=1}^{H}\frac{\sum_{i\in\mathcal{S}} p_{h,i}}{\sum_{i\in\mathcal{O}_h} p_{h,i}}, \tag{11}$$

where H is the number of core-attention heads and $\mathcal{O}_h$ contains the $k_t = \min(k, N_t)$ highest-probability tokens for head h. Thus, the metric measures how much attention mass the shared token set retains relative to each head's own best set of the same size. DSA also uses a shared selection rather than a separate optimum for each head, so its recall need not reach 100%. All methods are evaluated alongside the same native forward pass and share the dense-attention reference at each layer. For each context length, we average R equally over all layers (61 for DeepSeek-V3.2 and 78 for GLM-5.2), examples, and tasks.

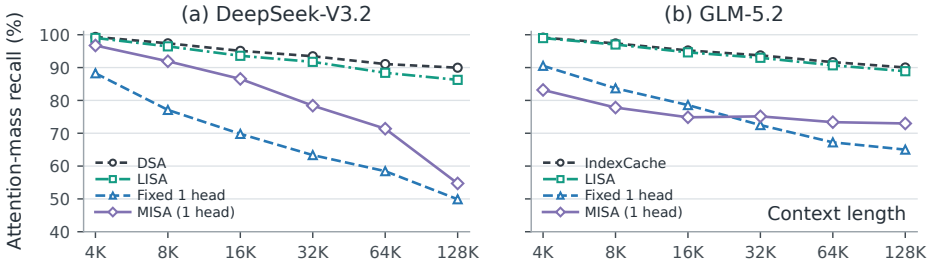

Figure 2: Attention-mass recall (%) on DeepSeek-V3.2 (left) and GLM-5.2 (right), with $k = 2048$ and native index reuse retained on GLM-5.2. Fixed 1 head uses head IDs 33 and 12, respectively.

We first compare LISA with the fixed single-head and query-dependent MISA ($h = 1$) controls to examine the value of head fusion (Figure 2). LISA achieves higher attention-mass recall than both controls at every tested context length on both models. At 128K, LISA reaches 86.26% on DeepSeek-V3.2 and 88.89% on GLM-5.2, compared with 49.89% and 65.01% for the fixed single-head control, and 54.72% and 72.98% for MISA ($h = 1$). These results support combining the weighted linear contributions of indexer heads to retain attention mass when scoring tokens with a fused query.

Although LISA outperforms both single-head controls, a gap to the respective native baselines remains at longer contexts (Figure 3). At 128K, LISA† increases recall from 86.26% to 89.41% on DeepSeek-V3.2 and from 88.89% to 89.88% on GLM-5.2, approaching the native baselines of 89.96% and 90.03%. This closes approximately 85% and 87% of LISA's recall gap to the respective native baselines. By retaining a broader candidate pool and applying the original multi-head scorer within it, LISA† allows the nonlinear corrections omitted by linear screening to influence the final selection. Despite higher recall, LISA† does not consistently outperform LISA on RULER.

We next examine candidate-pool sharing across layers with LISA‡ on DeepSeek-V3.2. At 128K, its recall is 87.24%, compared with 89.41% for LISA†, which constructs candidates independently at each layer. LISA‡ exceeds both LISA and IndexCache at all six context lengths. The shared pool reduces repeated full-context screening, while each layer's own scorer can adapt the final token selection within that pool.

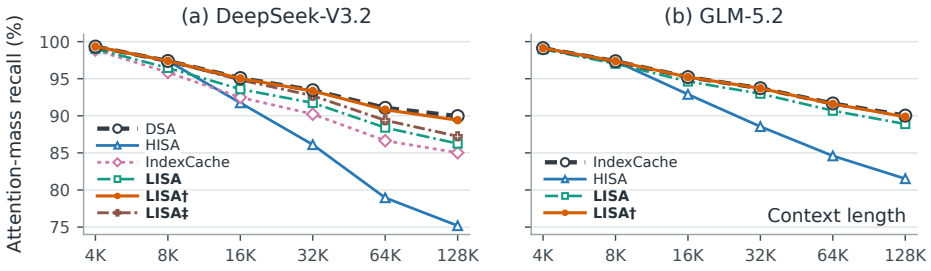

Figure 3: Dense-attention mass recall (%) on DeepSeek-V3.2 (left) and GLM-5.2 (right) across six context lengths. Full numerical results are provided in Table 4 in Appendix A.

4.5 Score Visualization

To examine the scoring information retained by head fusion, we visualize the original indexer score, its linear term, and the omitted per-head corrections. Figure 4 shows this decomposition for a single 32K LongBench-v2 (Bai et al., 2025) example on both models.

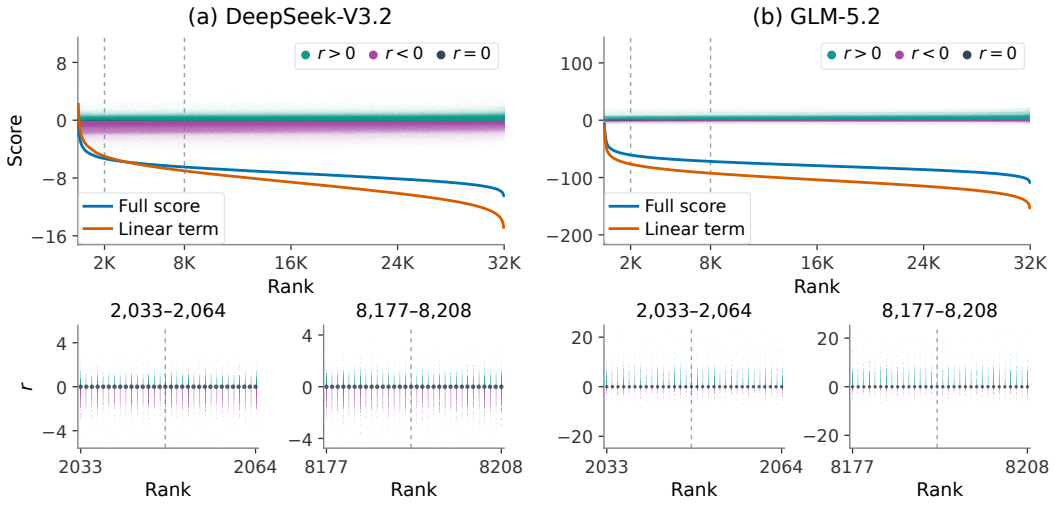

Figure 4: Score decomposition at the final prefill query, with tokens ranked by decreasing linear score within each layer. Curves show smoothed layer means; dots show per-head contributions r to $c_{t,s}$, with sizes at zero indicating counts within each panel. Bottom panels zoom around k and M.

In this example, the layer-averaged linear and full scores exhibit similar overall trends on both models. Of the per-head nonlinear corrections, 58.35% are zero on DeepSeek-V3.2 and 66.06% on GLM-5.2. For these head–token pairs, the linear term already equals the original weighted-ReLU contribution. Within the displayed local windows, positive and negative corrections partially cancel across heads, with stronger cancellation on DeepSeek-V3.2. These observations help explain why the head-fused linear score remains useful for token selection in LISA.

The remaining gap between the mean score curves varies with rank, so the nonlinear correction is not simply a constant score shift. Thus, the similar average trends do not imply identical token rankings, consistent with the recall gap between LISA and the native baselines in Section 4.4.

5 Conclusion

We presented LISA, a training-free indexer that exploits shared keys to fuse weighted query vectors across heads and directly selects tokens with the resulting linear score, removing the head-count factor from full-context scoring. With 64 indexer heads and 1M keys, LISA achieves a $6.12\times$ speedup over DSA for the complete selector. On DeepSeek-V3.2 with a 1M-token input, LISA achieves a $4.13\times$ speedup over DSA in single-request offline TTFT. Across 4K–128K contexts, LISA maintains RULER scores close to the respective native baselines on DeepSeek-V3.2 and GLM-5.2. Single-head comparisons show that LISA retains more attention mass than either fixed or query-dependent single-head selection. Candidate refinement in LISA† improves attention-mass recall, and LISA‡ reduces repeated screening through candidate-pool sharing while preserving layer-specific token selection.

Limitations and future work. LISA's linear score omits the nonlinear correction and need not preserve the original token ranking. Refinement in LISA† and LISA‡ cannot recover tokens excluded during candidate construction. We use pretrained indexers without additional training; whether training can improve selection with the linear score or support smaller candidate budgets remains to be studied. Our performance evaluation covers prefill indexing and single-request offline TTFT. Performance during decoding, under concurrent serving workloads, and on other hardware remains to be evaluated.

AI USE STATEMENT

In this work, we used generative AI tools to assist with translation and implementation of the proposed methods. We did not use these tools to develop the core methodological ideas, design the experiments, interpret the experimental results, or generate datasets. We also used these tools for literature search, reference formatting, and preparing plots from experimental data. The authors reviewed the translations, tested the AI-assisted code, checked the cited sources, and verified the figures against the experimental data. The authors take responsibility for the final manuscript, including its claims, code, figures, and experimental results.

REFERENCES

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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025. doi: 10.18653/v1/2025.acl-long.183. URL https://aclanthology.org/2025.acl-long.183/.

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

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

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

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

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. Technical report, DeepSeek-AI, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash/resolve/main/DeepSeek_V41_Tech_Report.pdf.

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

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

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, et al. MiniMax sparse attention. *arXiv preprint arXiv:2606.13392*, 2026.

Hong Liu, Yuan Cheng, Lin Niu, Yi Su, Yufei Xue, Anmin Liu, Guanghua Yu, and Jianchen Zhu. PIVOT: Efficient query-group indexing for token-level sparse attention. *arXiv preprint arXiv:2607.24593*, 2026. URL https://arxiv.org/abs/2607.24593v1.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, et al. MoBA: Mixture of block attention for long-context LLMs. *arXiv preprint arXiv:2502.13189*, 2025.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Yutao Sun, Yanqi Zhang, Li Dong, Jianyong Wang, and Furu Wei. You only index once: Cross-layer sparse attention with shared routing. *arXiv preprint arXiv:2606.06467*, 2026.

Yufei Xu, Fanxu Meng, Fan Jiang, Yuxuan Wang, Ruijie Zhou, Zhaohui Wang, Jiexi Wu, Zhixin Pan, Xiaojuan Tang, Wenjie Pei, Tongxuan Liu, Di Yin, Xing Sun, and Muhan Zhang. HISA: Efficient hierarchical indexing for fine-grained sparse attention. In *Conference on Language Modeling (COLM)*, 2026. URL https://arxiv.org/abs/2603.28458v4.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. HELMET: How to evaluate long-context language models effectively and thoroughly. In *International Conference on Learning Representations (ICLR)*, 2025.

Ziqi Yin, Jianyang Gao, Peiqi Yin, Jiangneng Li, and Gao Cong. LiteTopK: Exploiting the curse of dimensionality for a fused Indexer-TopK kernel in long-context sparse attention. *arXiv preprint arXiv:2607.11976*, 2026. URL https://arxiv.org/abs/2607.11976.

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

Wen Zan, Jiaqi Zhang, Jianchao Tan, Hong Liu, Cunguang Wang, Xiang Li, Duyue Ma, Guanyu Wu, Yifan Lu, Fengcun Li, Yerui Sun, Peng Pei, Yuchen Xie, and Xunliang Cai. LongCat Sparse Attention: Taming the lightning via streaming-aware hierarchical cross-layer indexing. *arXiv preprint arXiv:2608.01662*, 2026. URL https://arxiv.org/abs/2608.01662v2.

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/2605.07363v1.

A Detailed Attention-Mass Recall Results

Table 4 reports the full numerical results underlying the single-head comparisons in Figure 2 and the method comparisons in Figure 3. All results use the attention-mass recall metric in Eq. (11) and follow the evaluation protocol in Section 4.4.

Table 4: Attention-mass recall (%) on DeepSeek-V3.2 and GLM-5.2 across six context lengths. Avg. denotes the unweighted mean across the six context lengths. All methods on GLM-5.2 retain the native IndexCache schedule ($G = 4$). The fixed single-head controls use zero-based head IDs 33 and 12, respectively.

Model	Method	4K	8K	16K	32K	64K	128K	Avg.
DeepSeek-V3.2	DSA	**99.33**	**97.40**	**95.09**	**93.45**	**91.07**	**89.96**	**94.38**
	IndexCache ($G = 2$)	99.04	96.42	93.52	91.49	88.31	86.93	92.62
	IndexCache ($G = 4$)	98.86	95.86	92.49	90.22	86.65	85.02	91.52
	HISA	**99.33**	**97.40**	91.75	86.11	78.95	75.19	88.12
	LISA	99.03	96.46	93.59	91.74	88.41	86.26	92.58
	LISA†	**99.33**	**97.40**	95.04	93.35	90.82	89.41	94.22
	LISA‡ ($G = 4$)	**99.33**	**97.40**	94.83	92.71	89.40	87.24	93.48
	Fixed 1 head	88.27	77.10	69.80	63.34	58.45	49.89	67.81
	MISA (1 head)	96.73	91.92	86.55	78.40	71.38	54.72	79.95
GLM-5.2	IndexCache ($G = 4$)	**99.12**	**97.36**	**95.22**	**93.72**	**91.67**	**90.03**	**94.52**
	HISA	**99.12**	**97.36**	92.89	88.55	84.60	81.52	90.67
	LISA	98.98	97.02	94.63	92.98	90.69	88.89	93.87
	LISA†	**99.12**	**97.36**	**95.22**	93.71	91.60	89.88	94.48
	Fixed 1 head	90.54	83.69	78.58	72.48	67.22	65.01	76.25
	MISA (1 head)	83.18	77.82	74.86	75.14	73.37	72.98	76.23

B Detailed Algorithms and Implementation

This appendix specifies the selection procedures for LISA, LISA†, and LISA‡ using the notation of Section 3.

B.1 Single-Layer Selection

Algorithm 1 summarizes LISA and LISA† for a query with visible positions $\mathcal{V}_t$ and $N_t = |\mathcal{V}_t|$. Both variants form the fused query $\mathbf{g}_t$; LISA selects directly from its linear scores, while LISA† constructs candidates and then applies the original multi-head scorer. We denote the candidate construction in lines 5–23 of Algorithm 1 by BuildCandidates. Given the fused query, keys, visible positions, and budgets, this procedure returns Ω_t; Sections B.2 and B.3 provide the threshold and recovery details. After candidate construction, we evaluate Eq. (1) at each retained position and return the $k_t = \min(k, N_t)$ highest-scoring positions. The procedures below assume valid visibility intervals and finite computed scores.

B.2 Sampling and Threshold Estimation

For packed key indices $0, \ldots, n-1$, write each query's visible interval as $\mathcal{V}_t = [a_t, b_t)$. For a tile of up to 64 queries, let $[A, E)$ cover their visible intervals and let $J = \min(S, E - A)$, where S is the sampling budget. We use the deterministic stratified positions

$$p_i = A + \left\lfloor \frac{(2i+1)(E-A)}{2\max(J,1)} \right\rfloor, \qquad i = 0, \ldots, J-1. \tag{12}$$

Queries in the tile share these positions, but each query retains only the samples within its own visible interval; let S_t be its resulting sample count.

For $N_t > M$, we estimate the descending sample rank corresponding to the target fraction M/N_t as

$$r_t = \mathrm{clip}_{[1,S_t]}\left(\left\lfloor \frac{S_t M + \lfloor N_t/2 \rfloor}{N_t} \right\rfloor \right). \tag{13}$$

We partition the valid sampled score range into 256 equal-width bins, flooring the range width at 10^{-20} and clipping bin indices to $[0, 255]$. Accumulating bin counts from high to low scores identifies the boundary bin whose cumulative count first reaches or exceeds r_t. The initial threshold $\tau_t^{(0)}$ is the minimum actual sampled score in that bin.

Algorithm 1 Single-layer selection for LISA and LISA†

Input: Indexer queries $\{\mathbf{q}_{t,j}^I\}_j$, weights $\{w_{t,j}^I\}_j$, keys $\{\mathbf{k}_s^I\}_s$, and visible positions $\mathcal{V}_t$
Input: Variant $v \in \{\mathrm{LISA}, \mathrm{LISA}^\dagger\}$; budgets $k \le M \le C$
Output: Selected positions $\mathcal{T}_t \subseteq \mathcal{V}_t$ with $|\mathcal{T}_t| = k_t$

$k_t \leftarrow \min(k, N_t)$; $\mathbf{g}_t \leftarrow \sum_j w_{t,j}^I \mathbf{q}_{t,j}^I$
if $v = \mathrm{LISA}$ **then**
  return $\mathrm{TopK}(\{\mathbf{g}_t^\mathsf{T}\mathbf{k}_s^I \mid s \in \mathcal{V}_t\}, k_t)$
end if
if $N_t \le M$ **then**
  $\Omega_t \leftarrow \mathcal{V}_t$
else
  Estimate $\tau_t^{(0)}$ from sampled linear scores as in Section B.2
  Scan $\mathcal{V}_t$ and compute $\ell_{t,s} = \mathbf{g}_t^\mathsf{T}\mathbf{k}_s^I$ for each visible position
  Pack up to C IDs satisfying $\ell_{t,s} \ge \tau_t^{(0)}$ into P_t and count all matches in m_t
  if $m_t \notin [k_t, \min(C, N_t)]$ **then**
    Cache $\{\ell_{t,s} \mid s \in \mathcal{V}_t\}$ for this query
    $\Theta_t \leftarrow 2^{32}\,\mathrm{ord}_{32}(\tau_t^{(0)})$; initialize (L_t, U_t) as in Section B.3
    $i \leftarrow 0$
    while $m_t \notin [k_t, \min(C, N_t)]$ **do**
      Update (L_t, U_t) using Eq. (15)
      Choose Θ_t using the midpoint or first-iteration probe in Section B.3
      $m_t \leftarrow \sum_{s \in \mathcal{V}_t} \mathbf{1}[\kappa_{t,s} \ge \Theta_t]$; $i \leftarrow i + 1$
    end while
    Pack all IDs $s \in \mathcal{V}_t$ satisfying $\kappa_{t,s} \ge \Theta_t$ into P_t
  end if
  $\Omega_t \leftarrow P_t$
end if
$I_{t,s} \leftarrow \sum_{j=1}^{H^I} w_{t,j}^I\,\mathrm{ReLU}\big((\mathbf{q}_{t,j}^I)^\mathsf{T}\mathbf{k}_s^I\big)$ for all $s \in \Omega_t$
return $\mathrm{TopK}(\{I_{t,s} \mid s \in \Omega_t\}, k_t)$

When $N_t \le M$, the implementation uses $\tau_t^{(0)} = -\infty$, so every visible position becomes a candidate. The batched implementation still launches fusion, sampling, and scanning for this case. Our experiments use $S = 2048$, $M = 8192$, $C = 16384$, and $k = 2048$.

B.3 Candidate Compaction and Count Recovery

The full-context scan fuses linear scoring, threshold comparison, and candidate compaction, avoiding a full query-by-key score matrix on the normal path. The scan writes at most C token identifiers into P_t, while its counter m_t includes every visible position that passes the threshold. If $k_t \le m_t \le \min(C, N_t)$, this buffer contains the complete candidate set and can proceed to refinement. Otherwise, recovery adjusts the selection cutoff using the full visible context, because the identifiers written before the buffer fills are not ordered by score. The target M guides threshold estimation; a valid candidate count need not equal M.

Queries requiring recovery are processed in groups of at most 64, with their linear scores recomputed and cached once per group. To split a tied boundary group when necessary, we encode each computed FP32 score and its token position as

$$\kappa_{t,s} = 2^{32}\,\mathrm{ord}_{32}(\ell_{t,s}) + (n - s). \tag{14}$$

Here, ord_{32} maps FP32 values to unsigned integers in numerical order, with both signed zeros mapped to the same value. For the raw FP32 bits b after zero canonicalization, the encoding uses the bitwise complement of b for negative values and $b \oplus 2^{31}$ otherwise. Larger scores have larger keys, and smaller token indices have priority among equal scores. The initial cutoff is $\Theta_t = 2^{32}\,\mathrm{ord}_{32}(\tau_t^{(0)})$, with search bounds $L_t = 2^{32}\,\mathrm{ord}_{32}(-\infty)$ and $U_t = 2^{32}\,\mathrm{ord}_{32}(+\infty)$.

Each recovery iteration updates the bounds according to the current count:

$$(L_t, U_t) \leftarrow \begin{cases} (\max(L_t, \Theta_t + 1),\, U_t), & m_t > C, \\ (L_t,\, \min(U_t, \Theta_t - 1)), & m_t < k_t. \end{cases} \tag{15}$$

The next cutoff is normally the integer midpoint $L_t + \lfloor (U_t - L_t)/2 \rfloor$. On the first iteration for an overflowing query, we try a cutoff within the current boundary-score group:

$$\begin{aligned} c_t &= a_t + \max\left(1, \left\lfloor \frac{N_t M}{m_t} \right\rfloor\right) - 1, \\ \Theta_{\text{probe}} &= 2^{32} \left\lfloor \frac{\Theta_t}{2^{32}} \right\rfloor + (n - c_t). \end{aligned} \tag{16}$$

We use this probe only if it lies in the updated interval $[L_t, U_t]$; otherwise, we use the midpoint. After each update, we count positions satisfying $\kappa_{t,s} \geq \Theta_t$ from the cached scores and stop once the count is valid. We then regenerate the candidate buffer using this full score-and-position cutoff, including its tie-breaking component. CUDA conditional graph loops control the recovery groups and iterations without host polling inside the search. These operations enforce the candidate-count bounds; refinement still considers only the retained candidates and cannot recover positions excluded by linear screening.

B.4 Sharing Candidate Pools across Layers

Algorithm 2 constructs a candidate pool at the first layer a_r of each group $\mathcal{G}_r$ and refines it at every layer in that group, including a_r. Each layer supplies its own queries, keys, and head weights as it is executed, so its final selection can differ from those of the other layers. For DeepSeek-V3.2, we use zero-based layer groups $\{0\}, \{1, \ldots, G\}, \{G+1, \ldots, 2G\}$, and so on, with $G = 4$. Candidate positions and valid counts are retained for the same forward pass and corresponding query chunk until all layers in the group have used them. The pool storage is kept valid across selector workspace reuse and is released after the last member consumes it.

Algorithm 2 LISA‡ with shared candidate pools

Input: Consecutive layer groups $\{\mathcal{G}_r\}_r$, visible positions $\{\mathcal{V}_t\}_t$, and budgets $k \leq M \leq C$
Output: Layer-specific selected positions $\{\mathcal{T}_{t,l}\}_{t,l}$

for each group $\mathcal{G}_r$ in model order **do**
  for each layer $l \in \mathcal{G}_r$ in execution order **do**
    Obtain the current layer's indexer queries, keys, and head weights
    for each query t in the current batch **do**
      if $l = a_r$ **then**
        $\mathbf{g}_t^{(a_r)} \leftarrow \sum_j w_{t,j}^{I,(a_r)} \mathbf{q}_{t,j}^{I,(a_r)}$
        $\Omega_{t,r} \leftarrow \mathrm{BuildCandidates}(\mathbf{g}_t^{(a_r)}, \{\mathbf{k}_s^{I,(a_r)}\}_s, \mathcal{V}_t, k, M, C)$
      end if
      $I_{t,s}^{(l)} \leftarrow \sum_{j=1}^{H^I} w_{t,j}^{I,(l)} \,\mathrm{ReLU}\Big((\mathbf{q}_{t,j}^{I,(l)})^{\top} \mathbf{k}_s^{I,(l)}\Big)$ for all $s \in \Omega_{t,r}$
      $\mathcal{T}_{t,l} \leftarrow \mathrm{TopK}(\{I_{t,s}^{(l)} \mid s \in \Omega_{t,r}\}, k_t)$
    end for
    Use $\{\mathcal{T}_{t,l}\}_t$ in layer l's sparse attention and continue the model forward pass
  end for
  Release the candidate pools for $\mathcal{G}_r$
end for