LATENTINDEX: CROSS-LAYER SHARING WITH LAYER-SPECIFIC SELECTION FOR SPARSE ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse attention reduces core-attention computation, but its indexers still incur repeated selection work and per-layer key-cache storage. Reusing selected indices across layers reduces this overhead but constrains multiple layers to the same token set. We introduce LatentIndex, which extends the latent-sharing principle of Multi-head Latent Attention across indexer layers. Each layer group constructs a shared latent cache from its first layer's hidden states, while layer-specific scoring enables independent token selection. Absorbing key decoders into queries enables direct scoring of the shared cache without reconstructing historical per-layer keys. We develop training-free calibration and investigate a training-aware instantiation of this principle. To balance quality and computation, a hierarchical selection (HS) variant lets followers independently refine a shared candidate set proposed by the anchor. With four-layer sharing, LatentIndex reduces logical indexer-cache storage by 61.1% on DeepSeek-V3.2. Across DeepSeek-V3.2 and GLM-5, training-free LatentIndex improves head-wise attention-mass recall over IndexCache by up to 3.28 percentage points while maintaining RULER and LongBench performance close to native DSA. HS further achieves 2.30–2.72× decode indexer speedups over DSA across 8K–128K contexts, retaining most of LatentIndex's recall. LatentIndex offers a new perspective on cross-layer indexing: sharing continuous representations rather than discrete selections enables efficient reuse while preserving layer-specific token selection.

1 INTRODUCTION

Long-context processing is increasingly important for large language models. Web-based agents and software-engineering workflows require models to integrate information across multiple interaction steps and diverse sources (Nakano et al., 2021; Jimenez et al., 2024), while extended reasoning further increases context lengths. These workloads make attention efficiency critical to inference speed and serving capacity. Dense attention requires $\mathcal{O}(L^2)$ computation over a sequence of length L; during autoregressive decoding, each new query also attends to a growing history, increasing computation and KV-cache traffic.

Sparse attention reduces this cost by restricting the tokens or blocks that each query attends to (Child et al., 2019; Beltagy et al., 2020; Kitaev et al., 2020; Yuan et al., 2025; Lu et al., 2025). DeepSeek Sparse Attention (DSA) (DeepSeek-AI, 2025) implements fine-grained token selection through a lightning indexer, which scores historical tokens and selects the top-k positions for core attention. This reduces core-attention complexity to $\mathcal{O}(Lk)$ over the sequence. However, the indexer scans the full prefix at each layer, retaining quadratic prefill complexity and maintaining a separate key cache. Sparse core attention therefore leaves two sources of indexing overhead: repeated selection computation and per-layer key storage.

Cross-layer redundancy offers an opportunity to reuse indexing information. IndexCache (Bai et al., 2026) reports substantial overlap between nearby layers' top-k selections and exploits it by retaining indexers in a subset of Full layers while letting Shared layers reuse their selected indices. This reduces indexing computation but constrains multiple layers to the same token set, preventing followers from adapting their selections to their own queries and relevance distributions. This motivates our central question: *Can we exploit cross-layer redundancy while retaining layer-specific token selection?*

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.

Multi-head Latent Attention (MLA) (DeepSeek-AI, 2024) offers a useful design principle: a compact latent representation can be shared across attention heads while supporting head-specific attention. We extend this principle to indexer layers with **LatentIndex**. Each group of consecutive layers constructs one shared latent cache from the hidden states of its first layer, termed the anchor. Layer-specific decoders define effective indexer keys for all group members, including the anchor, while each layer retains its own scoring and top-k selection. At inference time, the decoders are absorbed into queries, enabling direct scoring of the shared cache without reconstructing historical per-layer keys. This design replaces separate indexer key caches with a shared representation while leaving the main-attention KV caches unchanged.

To further accelerate indexing while preserving layer-specific decisions, we introduce a *hierarchical selection* (HS) variant. The anchor scans the full prefix to form a broad candidate set, and each follower independently scores these candidates and selects its own top-k tokens. HS thus balances computational savings with layer-specific selection quality.

We instantiate LatentIndex in both training-free and training-aware settings. Training-free calibration uses offline ridge reduced-rank regression to jointly fit shared projections and layer-specific decoders, yielding a shared representation that serves the entire group. All fitted parameters remain fixed during inference. Training-aware adaptation begins with a frozen backbone, training each layer's indexer to match its dense-attention distribution. It then updates the backbone through sparse language modeling while continuing attention-based supervision of the indexers, without changing the main-attention architecture.

We evaluate LatentIndex in both settings. On DeepSeek-V3.2 and GLM-5, training-free LatentIndex improves head-wise attention-mass recall over IndexCache at every evaluated length from 4K to 128K, by up to 3.28 percentage points. Its aggregate RULER and LongBench scores exceed IndexCache's on both models and remain close to native DSA's. With four-layer sharing on DeepSeek-V3.2, LatentIndex reduces logical indexer-cache storage by 61.1%. HS retains most of its recall while achieving 2.30–2.72$\times$ decode indexer speedups over DSA across 8K–128K, measured across all 61 indexers. Compared with IndexCache, HS improves average recall, RULER, and LongBench scores through additional layer-specific candidate scoring. Training experiments on DeepSeek-V2-Lite further show higher warmup recall and generally lower sparse-training language-modeling loss than IndexCache.

Our contributions are threefold:

- We introduce LatentIndex, bringing MLA's latent-sharing principle to sparse-attention indexers across layers. Shared latent caching reduces indexer-cache storage, while query-side decoder absorption enables direct scoring with layer-specific token selection.
- We develop training-free calibration and investigate training-aware adaptation. We further introduce hierarchical selection to reduce repeated full-prefix scoring through shared candidates and layer-specific refinement.
- We demonstrate improved selection quality over index reuse, downstream performance close to native DSA, and a quality–latency trade-off through hierarchical selection. Training experiments further show that shared latent indexers can be learned while maintaining language-model loss close to native DSA.

2 Related Work

Sparse Attention. Sparse attention reduces attention cost by restricting the positions accessed by each query through fixed patterns, local/global connectivity, or content-dependent selection (Child et al., 2019; Beltagy et al., 2020; Kitaev et al., 2020). Recent methods include Native Sparse Attention (NSA), MoBA, and MiniMax Sparse Attention, which combine selective access with efficient attention execution (Yuan et al., 2025; Lu et al., 2025; Lai et al., 2026). DeepSeek Sparse Attention (DSA) (DeepSeek-AI, 2025) uses a lightning indexer to select tokens for the main attention computation. Although this reduces the main attention cost, the indexer still scores the full causal prefix for each query, retaining quadratic scoring complexity during prefill.

Efficient Indexing for Sparse Attention. Existing approaches improve indexing efficiency through cross-layer reuse, restricted search, and reduced representation or head costs. Index-

Cache (Bai et al., 2026) lets subsequent layers reuse an anchor layer's top-k indices, while DeepSeek-V4.1-Flash (DeepSeek-AI, 2026) includes a Reuse mode for the available main KV cache and its latest selection. SparDA (Fu et al., 2026) predicts the KV blocks needed by the next layer, enabling prefetching to overlap with current-layer execution. HISA (Xu et al., 2026b) selects candidate blocks before token-level refinement, and PIVOT (Liu et al., 2026) amortizes full-prefix scoring across nearby queries through a proxy query. Other designs select over compressed representations (Xu et al., 2026a), dynamically activate indexer heads (Zhou et al., 2026), or use fewer query heads (Qiu et al., 2026). LatentIndex explores cross-layer representation sharing as a basis for efficient indexing while preserving layer-specific token selection.

Representation Sharing and Compression. Sharing continuous representations offers another route to reducing attention overhead. CLA (Brandon et al., 2024) and MLKV (Zuhri et al., 2025) share key/value heads across layers, while YOCO (Sun et al., 2024) constructs global KV representations for reuse across its cross-decoder layers. KVSharer (Yang et al., 2024) explores post-hoc layer-wise cache sharing. Beyond direct reuse, xKV (Chang et al., 2025) jointly factorizes grouped-layer KV caches into a shared low-rank subspace, and CommonKV (Wang et al., 2025) factorizes adjacent-layer projection weights to facilitate latent-cache merging. Dimensional compression also supports efficient attention and retrieval: MLA (DeepSeek-AI, 2024) combines latent KV storage with query-side absorption of content-key projections, while Loki (Singhania et al., 2024) exploits low-dimensional key structure for token selection. LatentIndex extends representation sharing from main-attention KV representations to the key representations maintained across indexer layers.

3 METHOD: LATENTINDEX

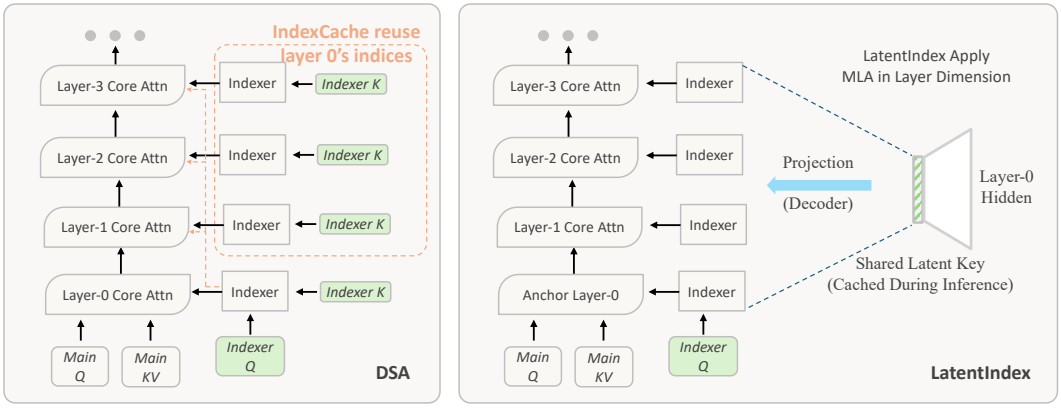


Figure 1: DSA uses per-layer indexer key caches and token selections; IndexCache reuses the anchor's selections (orange dashed paths). LatentIndex shares one latent cache derived from anchor hidden states, with layer-specific decoders defining layer-specific keys. At inference, decoders are absorbed into queries for direct cache scoring and independent token selection, without reconstructing historical keys. Queries are layer-specific but are illustrated only once for visual clarity.

LatentIndex (Figure 1) extends MLA's latent-sharing principle to the layer dimension of sparse-attention indexers, sharing key representations while retaining layer-specific token selection. Each group stores a shared latent cache, while layer-specific decoders define the effective keys used by each layer. At inference, these decoders are absorbed into queries, allowing each layer to score the shared cache directly without reconstructing historical keys. We first describe the shared representation and its scoring procedure, then explain how to obtain the parameters through offline calibration or training and how hierarchical selection reduces repeated full-prefix scoring.

3.1 SHARED KEY REPRESENTATIONS WITH LAYER-SPECIFIC INDEXERS

Notation. We partition indexer layers into consecutive groups $\{\mathcal{G}_g\}$ of at most G layers, with the first layer $a_g = \min \mathcal{G}_g$ serving as the anchor. Let $\mathbf{x}_t^{(\ell)}$ denote the hidden state of token t at layer

ℓ after input RMSNorm, represented as a column vector. The native indexer key and shared latent have dimensions $d_I = d_R + d_N$ and $d_C = r_R + r_N$, respectively, where the subscripts R and N indicate RoPE and NoPE components. Each indexer uses H_I query heads.

Shared Latent Cache. To replace separate per-layer key caches, each group constructs one shared representation per token from the anchor hidden state. This representation has a RoPE branch that encodes position and a NoPE branch. For token position t, we compute

$$\mathbf{z}_t^{(g),N} = \mathbf{P}_{g,N}^\top \mathbf{x}_t^{(a_g)}, \qquad \mathbf{z}_t^{(g),R} = \mathbf{Q}_t \mathbf{P}_{g,R}^\top \mathbf{x}_t^{(a_g)}, \tag{1}$$

where $\mathbf{P}_{g,N} \in \mathbb{R}^{d_{\text{model}} \times r_N}$ and $\mathbf{P}_{g,R} \in \mathbb{R}^{d_{\text{model}} \times r_R}$ are group-specific projections, and $\mathbf{Q}_t \in \mathbb{R}^{r_R \times r_R}$ applies rotary position encoding in the latent space. The two branches are concatenated and cached once per group:

$$\mathbf{c}_t^{(g)} = \begin{bmatrix} \mathbf{z}_t^{(g),R} \\ \mathbf{z}_t^{(g),N} \end{bmatrix} \in \mathbb{R}^{d_C}. \tag{2}$$

The two branches form a single cache used by every group member. Each layer interprets this shared representation through its own decoder, rather than directly reusing the anchor's native key.

Layer-specific Key Reconstruction. To let a shared cache support different relevance scores across layers, each layer uses its own decoder to map the latent representation to effective indexer keys.

For each $\ell \in \mathcal{G}_g$, layer-specific decoders define the effective key components:

$$\widehat{\mathbf{k}}_t^{(\ell),R} = \mathbf{D}^{(\ell),R} \mathbf{z}_t^{(g),R}, \qquad \widehat{\mathbf{k}}_t^{(\ell),N} = \mathbf{D}^{(\ell),N} \mathbf{z}_t^{(g),N} + \mathbf{b}^{(\ell)}, \tag{3}$$

where $\mathbf{D}^{(\ell),R} \in \mathbb{R}^{d_R \times r_R}$, $\mathbf{D}^{(\ell),N} \in \mathbb{R}^{d_N \times r_N}$, and $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_N}$. Because the rotary latent is already position-encoded when cached, its decoder must preserve the native rotary structure. We therefore require decoding the rotated latent to give the same result as rotating the decoded key:

$$\mathbf{R}_t \mathbf{D}^{(\ell),R} = \mathbf{D}^{(\ell),R} \mathbf{Q}_t, \tag{4}$$

for all t, where $\mathbf{R}_t \in \mathbb{R}^{d_R \times d_R}$ is the native rotary map. This condition makes decoding before native rotation equivalent to decoding after latent rotation. We enforce it through complex linear combinations within each rotary frequency, without cross-frequency mixing or a rotary bias (Appendix A.2).

Combining both components gives

$$\widehat{\mathbf{k}}_t^{(\ell)} = \mathbf{D}^{(\ell)} \mathbf{c}_t^{(g)} + \mathbf{b}_{\text{ext}}^{(\ell)}, \qquad \mathbf{D}^{(\ell)} = \begin{bmatrix} \mathbf{D}^{(\ell),R} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}^{(\ell),N} \end{bmatrix}, \qquad \mathbf{b}_{\text{ext}}^{(\ell)} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b}^{(\ell)} \end{bmatrix}, \tag{5}$$

with $\mathbf{D}^{(\ell)} \in \mathbb{R}^{d_I \times d_C}$. These effective keys define layer-specific scoring. Note that during actual inference, the decoders are absorbed into the queries, avoiding explicit key reconstruction.

Query-side Decoder Absorption. Explicitly reconstructing historical keys would introduce additional computation and memory overhead, but indexing only requires their dot products with the current queries. As in MLA, we absorb each layer's key decoder into its queries to score the shared cache directly. Let $\mathbf{q}_{s,h}^{(\ell)} \in \mathbb{R}^{d_I}$ denote the native indexer query at position s and head h, with native RoPE already applied to its rotary component. Using the decoder and bias defined above, we compute

$$\widetilde{\mathbf{q}}_{s,h}^{(\ell)} = \left(\mathbf{D}^{(\ell)}\right)^\top \mathbf{q}_{s,h}^{(\ell)} \in \mathbb{R}^{d_C}, \qquad \beta_{s,h}^{(\ell)} = \left(\mathbf{q}_{s,h}^{(\ell),N}\right)^\top \mathbf{b}^{(\ell)}, \tag{6}$$

where $\mathbf{q}_{s,h}^{(\ell),N}$ is the query's NoPE component. For any historical position $t \le s$, the query–key dot product satisfies

$$\left(\mathbf{q}_{s,h}^{(\ell)}\right)^\top \widehat{\mathbf{k}}_t^{(\ell)} = \left(\widetilde{\mathbf{q}}_{s,h}^{(\ell)}\right)^\top \mathbf{c}_t^{(g)} + \beta_{s,h}^{(\ell)}. \tag{7}$$

Thus, each decoder is applied once per query head, and the resulting query scores the shared cache without reconstructing historical keys. The bias contribution $\beta_{s,h}^{(\ell)}$ is shared across historical positions but must be added before the indexer's ReLU. This reformulation is exact for the effective keys in exact arithmetic.

Layer-specific Scoring and Selection. Each layer uses its own head weights $w_{s,h}^{(\ell)}$ and indexer scaling to score the shared cache over valid causal positions $\mathcal{V}_s$:

$$u_{s,t}^{(\ell)} = \sum_{h=1}^{H_I} w_{s,h}^{(\ell)} \operatorname{ReLU}\left(\left(\widetilde{\mathbf{q}}_{s,h}^{(\ell)} \right)^{\top} \mathbf{c}_t^{(g)} + \beta_{s,h}^{(\ell)} \right), \qquad t \in \mathcal{V}_s. \tag{8}$$

Each layer independently selects

$$\mathcal{S}_s^{(\ell)} = \operatorname{TopK}_{t \in \mathcal{V}_s} \left(u_{s,t}^{(\ell)}, \min(k, |\mathcal{V}_s|) \right), \tag{9}$$

which its main attention uses with its own KV cache. Thus, layers share the latent cache while retaining layer-specific scores and token selections.

3.2 Training-Free LatentIndex

Offline Calibration. We instantiate LatentIndex in a frozen pretrained model through offline calibration of the shared projections and layer-specific decoders. For each group, we collect the anchor hidden states and the native indexer keys of all group members at matching token positions. We treat the group members' keys as joint targets and use ridge reduced-rank regression Izenman (1975) to jointly fit a shared projection from the anchor hidden states and a separate decoder for each layer. The objective minimizes key-reconstruction error under the prescribed latent-rank constraints, with ridge regularization on the resulting mappings.

We fit the NoPE and RoPE branches separately. For NoPE, we concatenate the group members' target keys and fit a mapping of rank at most r_N with an output bias. For RoPE, we fit a bias-free complex reduced-rank mapping independently at each native frequency. Same-frequency decoding preserves compatibility with positional rotation. Both branches admit closed-form solutions, requiring no gradient-based optimization or updates to pretrained weights. The fitted projections generate the shared latent cache, and the fitted decoders define the query transformations described above. All fitted parameters remain fixed during inference. Appendix A provides more details.

Hierarchical Selection. To further accelerate indexing while preserving layer-specific token selection, we restrict follower layers to a shared candidate set, leaving the latent representation and query-side decoder absorption unchanged. For each query position s, the anchor a_g scores all valid causal positions $\mathcal{V}_s$ using Eq. equation 8 and forms a candidate set $\mathcal{C}_s^{(g)}$ from the top-B positions, where $B \geq k$. The anchor selects its own top-k positions under the same scores. Each follower $\ell \in \mathcal{G}_g \setminus \{a_g\}$ independently rescores these candidates using its layer-specific scorer and selects its own top-k set $\mathcal{S}_s^{(\ell)}$. Both budgets are capped by $|\mathcal{V}_s|$, and each layer uses its selected positions with its own main-attention KV cache. Thus, each follower scores at most B positions per query while retaining layer-specific selection within the shared candidate set.

3.3 Training-Aware LatentIndex

We also study training LatentIndex on DeepSeek-V2-Lite. Training proceeds in two stages: dense-attention warmup learns the indexers with a frozen backbone, followed by sparse training that adapts the backbone and indexers using their selected routes.

Training instantiation. In this instantiation, the anchor hidden state supplies both the shared key representation and the inputs to all layer-specific query and head-weight projections. Each layer uses its own NoPE decoder, query projection, and head-weight projection to compute relevance scores and independently select tokens. On DeepSeek-V2-Lite, we partition the 27 layers into nine consecutive groups of three, with top-2048 selection. For this training instantiation, we apply LayerNorm to the projected NoPE latent in Eq. equation 1, use bias-free NoPE decoders, and fix the rotary decoder to the identity. Because all routing inputs are available at the anchor, the group's routes can be computed at its entrance and then consumed by their respective layers during sparse execution.

Dense-attention warmup. We first freeze the pretrained backbone and train the indexers to match each layer's dense attention. For each query, the teacher distribution averages the core-attention

probabilities across heads, while the indexer distribution applies softmax to its scores over the full causal prefix. Training minimizes the teacher-to-indexer KL divergence over valid prediction positions. Each layer's dense-attention distribution supervises its LatentIndex scoring distribution. In IndexCache, all three layer-specific teacher distributions supervise the anchor indexer. The method-specific loss normalization is detailed in Appendix D.2. This stage learns token-selection scores from dense attention while leaving the backbone and its dense-attention computation unchanged.

Sparse training. Starting from each method's warmup checkpoint, we activate its top-2048 selections and train the backbone with next-token language modeling, retaining the main-attention architecture. The indexers continue to receive KL supervision, now using each layer's sparse-attention probabilities, with both distributions normalized over the selected positions. Teacher targets and indexer inputs are detached from the backbone, and the top-k operation does not pass language-modeling gradients to the selector. Additional training details are provided in Appendix D.1.

3.4 Storage and Computation

Indexer cache storage. For a group of m layers, we store one d_C-dimensional latent instead of m separate d_I-dimensional keys, giving a payload ratio of $d_C/(md_I)$. Including FP8 scales and the independent native layer, our DeepSeek-V3.2 configuration reduces logical indexer-cache storage by 61.1%. The main-attention KV cache remains unchanged.

Fixed parameters and computation. Projection and decoder parameters are independent of context length. We generate the shared latent once per group and absorb each layer's decoder into its queries. Without hierarchical selection, each layer independently scores the full causal history, requiring $O(H_I L d_C)$ work per query and $O(H_I L^2 d_C)$ over prefill. With candidate budget B, hierarchical selection reduces the number of scored positions per query in a group of m layers from mL to $L + (m-1)\min(B, L)$. Appendix B details storage accounting and inference implementation.

4 Experiments

4.1 Experimental Setup

We compare native DSA, IndexCache, and LatentIndex on DeepSeek-V3.2 (DeepSeek-AI, 2025) and GLM-5 (GLM-5 Team, 2026). Grouped methods use identical layer boundaries with $G = 4$ unless otherwise stated, and all methods select up to 2048 tokens per query. Training-free LatentIndex uses offline calibration and segmented FP8 inference, with calibration data disjoint from evaluation data. We evaluate token-selection quality using attention-mass recall and downstream performance using RULER (Hsieh et al., 2024) and LongBench (Bai et al., 2023). RULER covers six context lengths from 4K to 128K, while LongBench reports six task categories. Full configurations and evaluation protocols are provided in Appendix C.

4.2 Training-Free Token-Selection Quality

Metric. We measure how well an indexer's selected tokens capture the importance assigned by core attention. Attention-mass recall directly measures how well the selected tokens preserve the attention mass prioritized by core attention at a fixed token budget. For a fixed query and layer, let $p_{h,i}$ denote core-attention head h's dense attention probability over the full causal prefix, computed using the native attention scale. Let S be the indexer's selected set of $K = \min(2048, N)$ tokens for prefix length N, and let $T_h = \mathrm{TopK}_i(p_{h,i}, K)$ be head h's oracle selection. Head-wise attention-mass recall is

$$R_{\text{head}} = \frac{1}{H}\sum_{h=1}^{H} \frac{\sum_{i\in S} p_{h,i}}{\max\left(\sum_{i\in T_h} p_{h,i}, \epsilon\right)}, \qquad \epsilon = 10^{-8}, \tag{10}$$

where H is the number of core-attention heads. This metric averages the fraction of each head's oracle attention mass captured at the same token budget, giving all heads equal weight. Sampling and aggregation details are in Appendix C.

Table 1: Head-wise attention-mass recall on a common native DSA trajectory for each model.

Method	DeepSeek-V3.2						GLM-5					
	4K	8K	16K	32K	64K	128K	4K	8K	16K	32K	64K	128K
DSA	**99.33**	**97.40**	**95.09**	**93.48**	**91.10**	**90.00**	**99.45**	**98.33**	**97.21**	**96.37**	**95.16**	**94.30**
IndexCache	98.86	95.86	92.49	90.26	86.68	85.06	99.16	97.49	95.78	94.65	92.79	91.55
LatentIndex	99.21	96.97	94.34	92.40	89.61	88.34	99.32	97.96	96.65	95.67	94.22	93.12
Δ vs. IndexCache	+0.35	+1.11	+1.85	+2.14	+2.93	+3.28	+0.16	+0.47	+0.87	+1.02	+1.43	+1.57

Table 2: RULER performance across context lengths. All grouped methods use $G = 4$.

Model	Method	4K	8K	16K	32K	64K	128K	Avg.
DeepSeek-V3.2	DSA	97.21	96.84	**97.51**	**97.46**	95.89	**94.30**	**96.54**
	IndexCache	**97.28**	**96.94**	**97.51**	97.18	**95.91**	93.59	96.40
	LatentIndex	97.18	96.92	97.35	97.42	95.85	94.22	96.49
GLM-5	DSA	**98.07**	**97.57**	**97.69**	**97.41**	97.40	96.74	**97.48**
	IndexCache	97.73	97.42	97.51	96.92	95.70	90.15	95.91
	LatentIndex	97.89	**97.57**	97.64	97.36	**97.42**	**96.81**	97.45

Results. LatentIndex outperforms IndexCache at every evaluated length on both models, with gains increasing toward longer contexts (Table 1). At 128K, it improves recall by 3.28 percentage points on DeepSeek-V3.2 and 1.57 points on GLM-5. These results support sharing continuous representations while retaining layer-specific selection to preserve attention mass.

4.3 Training-Free Downstream Performance

We further evaluate each method end-to-end, where its token selections affect subsequent hidden states and generation.

RULER. LatentIndex closely preserves native DSA performance, with average scores only 0.05 and 0.03 points lower on DeepSeek-V3.2 and GLM-5, respectively (Table 2). It improves over IndexCache in aggregate on both models. On DeepSeek-V3.2 at 128K, it outperforms IndexCache by 0.63 points and remains 0.08 below DSA. The largest gain over IndexCache occurs on GLM-5 at 128K, where LatentIndex reaches 96.81, compared with 90.15 for IndexCache and 96.74 for DSA.

LongBench. Table 3 covers single-document QA (SQA), multi-document QA (MQA), summarization (Sum), few-shot learning (FS), synthetic retrieval (Syn), and code completion (Code). LatentIndex improves the average score over IndexCache by 0.73 points on DeepSeek-V3.2 and 0.30 points on GLM-5, while remaining close to native DSA. LatentIndex improves all six categories relative to IndexCache on DeepSeek-V3.2, with the largest gain in few-shot learning. On GLM-5, it improves three of the six categories, with the largest gain in synthetic retrieval.

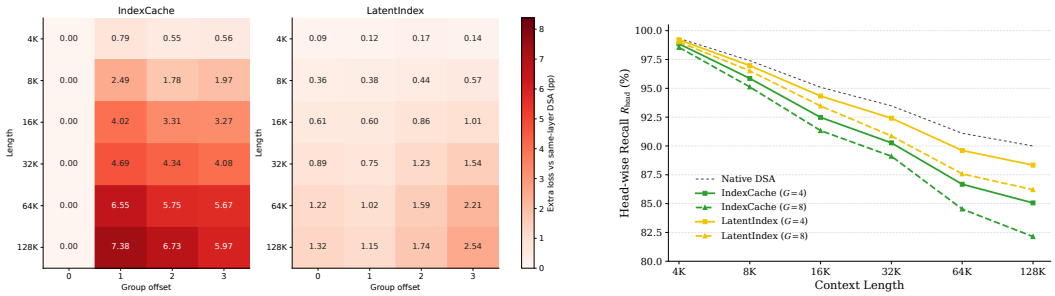
4.4 Analysis of Cross-Layer Sharing

Selection quality across layers within a group. We measure additional recall loss relative to each layer's native DSA selection: $E_m^{(\ell)} = R_{\text{head}}^{(\ell)}(S_{\text{DSA}}^{(\ell)}) - R_{\text{head}}^{(\ell)}(S_m^{(\ell)}), \quad m \in \{\text{IndexCache}, \text{LatentIndex}\}$. Within each four-layer group, offset 0 denotes the anchor and offsets 1–3 the followers. IndexCache preserves the anchor's native selection but incurs an immediate recall loss when that selection is reused by followers (Figure 2a). This loss grows with context length at every follower offset, reaching 5.97–7.38 percentage points at 128K. LatentIndex reduces recall loss at every follower offset and context length. At 128K, average follower loss decreases by 73%, from 6.69 to 1.81 percentage points, while anchor loss is 1.32 points. These results show that layer-specific selection mitigates the loss from cross-layer sharing, particularly at long contexts.

Effect of sharing group size. Figure 2b compares $G = 4$ and $G = 8$ with a fixed 192-dimensional latent. Larger groups reduce recall for both methods, particularly at long contexts, while LatentIn-

Table 3: LongBench performance across six task categories.

Model	Method	SQA	MQA	Sum	FS	Syn	Code	Avg.
DeepSeek-V3.2	DSA	**52.05**	**53.79**	**22.42**	**61.87**	68.17	73.12	55.24
	IndexCache	51.37	53.25	22.24	59.78	67.91	73.11	54.61
	LatentIndex	52.00	53.74	22.27	61.06	**68.78**	**74.18**	**55.34**
GLM-5	DSA	53.09	**56.02**	24.40	68.45	**76.50**	72.09	**58.42**
	IndexCache	53.01	55.67	**24.44**	**69.28**	73.33	**72.11**	57.97
	LatentIndex	**53.31**	55.71	24.35	68.58	76.33	71.34	58.27



(a) Recall loss by group offset

(b) Effect of group size

Figure 2: Cross-layer sharing on DeepSeek-V3.2: (a) additional R_{head} loss relative to native DSA by group offset ($G = 4$); (b) R_{head} across context lengths for $G = 4$ and $G = 8$, with native DSA as a reference.

dex consistently outperforms IndexCache at either group size. At 128K, LatentIndex with $G = 8$ achieves 86.21 recall, exceeding IndexCache with $G = 8$ by 4.07 percentage points and even IndexCache with $G = 4$ by 1.15 points. Increasing G from 4 to 8 also reduces LatentIndex's shared caches from 15 to 8 and its logical indexer-cache storage from 3132 to 1732 bytes per token, supporting larger sharing groups while retaining higher selection quality than index reuse.

4.5 Hierarchical Selection: Quality–Efficiency Trade-off

We evaluate LatentIndex with hierarchical selection (HS) on DeepSeek-V3.2 using a fixed candidate budget of 8192. Decode timing covers complete indexer forward calls across all 61 layers, including candidate construction and TopK but excluding main attention and other model components; timing details are in Appendix C.4. HS (Table 4) achieves $R_{\text{head}} = 93.19$, compared with 93.48 for full-prefix LatentIndex, with RULER and LongBench scores of 96.58 and 54.81, respectively. It exceeds IndexCache on all three aggregate quality metrics and achieves 2.30–2.72× decode indexer speedups over native DSA across 8K–128K. Relative to IndexCache, its additional indexer computation enables layer-specific refinement within shared candidates.

4.6 Training-Aware Evaluation

Protocol. We compare DSA, IndexCache, and training-aware LatentIndex on DeepSeek-V2-Lite at 64K, using eight indexer heads, top-2048 selection, and $G = 3$ for grouped methods. All methods complete 400 steps of indexer-only dense warmup, followed by sparse training that updates both the backbone and indexers. Sparse runs use matching samples at each step. Training configurations, method-specific loss normalization, and diagnostics are detailed in Appendix D.

Warmup selection quality. Figure 3 (a) reports matched-step Recall@2048 differences from DSA. The metric measures captured attention mass relative to each head's top-2048 oracle, averaged over sampled queries and all 27 layers; diagnostic scaling and sampling are detailed in Appendix D.3. After an initial widening, LatentIndex's gap to DSA narrows and stabilizes, whereas IndexCache's gap generally grows before leveling off. In later warmup steps, LatentIndex remains approximately

Table 4: Hierarchical selection on DeepSeek-V3.2. Latency is aggregated over all 61 indexers in ms per decode token; parentheses show speedup over native DSA. Recall and RULER are averaged over six context lengths, and LongBench over six task categories.

	Quality			Decode indexer latency (ms) ↓				
Method	R_{head} ↑	RULER ↑	LongBench ↑	8K	16K	32K	64K	128K
Native DSA	**94.40**	96.54	**55.24**	48.70	52.06	48.28	48.43	49.60
IndexCache	91.54	96.40	54.61	**14.31**	**13.29**	**14.35**	**14.28**	**13.40**
LatentIndex + HS	93.19	**96.58**	54.81	19.89 (2.45×)	19.13 (2.72×)	20.55 (2.35×)	21.10 (2.30×)	20.48 (2.42×)



(a) Warmup Recall@2048 (↑)

(b) Sparse-training LM loss (↓)

Figure 3: Training dynamics on DeepSeek-V2-Lite at 64K: (a) warmup Recall@2048 and (b) sparse-training LM loss. Values are method minus DSA at matched steps; the dashed line marks zero difference. Faint and bold curves show raw and EMA-smoothed differences, respectively.

1.3–1.5 percentage points below DSA, compared with 4.4–4.7 for IndexCache. These results show that LatentIndex learns selections more closely aligned with core attention than index reuse.

Sparse-training behavior. Figure 3(b) shows matched-step differences in training next-token cross-entropy. LatentIndex generally maintains lower LM loss than IndexCache and fluctuates closely around the DSA baseline. Together with its higher warmup recall, this indicates that shared latent indexing preserves selection quality while supporting sparse language-model training with loss close to DSA.

5 Conclusion

LatentIndex enables sparse-attention indexers to share continuous key representations across layers while retaining layer-specific token selection. Training-free results on DeepSeek-V3.2 and GLM-5 show higher attention-mass recall than index reuse and downstream performance close to native DSA, with a 61.1% reduction in logical indexer-cache storage on DeepSeek-V3.2. The recall advantage is particularly pronounced for followers at long contexts and persists with larger sharing groups. Hierarchical selection reduces repeated full-prefix scoring through shared candidates, achieving decode indexer speedups while retaining layer-specific refinement. Training experiments provide complementary evidence that shared latent indexers can be learned while maintaining language-model loss close to DSA. Together, these results support cross-layer representation sharing as a way to reduce indexer storage while preserving selection quality, with hierarchical selection providing additional computational savings.

Limitations and future work.

Our efficiency evaluation focuses on decode indexer latency at batch size one, with a fixed candidate budget for hierarchical selection. Future work will examine how batch size and candidate budget affect the quality–latency trade-off and explore adapting the budget to context length.

AI Use Statement

Generative AI tools were used to aid and polish the writing of this manuscript, including grammar and clarity checks. All final content was reviewed and approved by the authors.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiawei Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. LongBench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

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.

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

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. Reducing transformer key-value cache size with cross-layer attention. *arXiv preprint arXiv:2405.12981*, 2024.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, Xilai Dai, Huiqiang Jiang, Yucheng Li, Luis Ceze, Kai-Chiang Wu, and Mohamed S. Abdelfattah. xKV: Cross-layer SVD for KV-cache compression. *arXiv preprint arXiv:2503.18893*, 2025. ICML 2026. Compresses the MLA latent cache of DeepSeek models via cross-layer aligned singular vectors.

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

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

DeepSeek-AI. DeepSeek-V3.2 technical report. *Technical Report*, 2025. Introduces DeepSeek Sparse Attention (DSA) with lightning indexer.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026.

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

GLM-5 Team. GLM-5: From vibe coding to agentic engineering, 2026. URL `https://arxiv.org/abs/2602.15763`.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Agrawal, Dima Rekesh, Fei Xu, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Alan Julian Izenman. Reduced-rank regression for the multivariate linear model. *Journal of multivariate analysis*, 5(2):248–264, 1975.

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? *International Conference on Learning Representations (ICLR 2024)*, 2024.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *International Conference on Learning Representations (ICLR)*, 2020.

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.

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

Reiichiro Nakano, Jacob Hilton, Asya Balwit, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vinay Kosaraju, William Saunders, et al. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, et al. On the design of qwen3. 8-next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *Advances in Neural Information Processing Systems*, 37: 16692–16723, 2024.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*, 2024.

Yixuan Wang, Haoyu Qiao, Lujun Li, Qingfu Zhu, and Wanxiang Che. CommonKV: Compressing KV cache with cross-layer parameter sharing. *arXiv preprint arXiv:2508.16134*, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026a.

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*, 2026b.

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. KVSharer: Efficient inference via layer-wise dissimilar KV cache sharing. *arXiv preprint arXiv:2410.18517*, 2024.

Jingze Yuan et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025. ACL 2025 Best Paper.

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.

Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. Mlkv: Multi-layer key-value heads for memory efficient transformer decoding. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5531–5540, 2025.

A Offline Construction of LatentIndex

We construct training-free LatentIndex by jointly fitting a shared projection and layer-specific decoders to each group's native indexer keys, including those of the anchor. We fit the NoPE and RoPE branches separately using ridge reduced-rank regression.

A.1 Calibration Data and Targets

We freeze the pretrained backbone and collect features along its native DSA trajectory. At matching token positions, we record the anchor hidden states after input RMSNorm and each group member's native indexer keys after key normalization but before RoPE, Hadamard rotation, and quantization. We fit only key-reconstruction targets.

We draw calibration data from FineWeb-Edu, GitHub source code, OpenMathInstruct-2, SmolTalk, and PG19, covering general text, code, mathematics, dialogue, and long documents. Each domain contributes eight 8K and two 32K training requests, plus two 8K and one 32K validation requests. This yields 50 training and 15 validation requests, totaling 655,360 and 245,760 tokens under the DeepSeek-V3.2 tokenizer. General-text, mathematics, and dialogue requests may concatenate examples; each code or book request uses a single repository or book. We check source identities and text overlap across splits and keep calibration data separate from downstream evaluation. GLM-5 follows the same construction protocol.

All training prompt positions contribute to fitting. We use validation data to select regularization strengths and retain the training-fitted parameters without refitting on the combined splits.

A.2 RoPE-Compatible Decoders

Same-frequency decoding. For each native rotary frequency f, we construct r_f complex latent sources $v_{t,f,j}$ from the anchor hidden state and reconstruct layer ℓ's pre-RoPE key coordinate as

$$\widehat{k}^{(\ell),R}_{t,f} = \sum_{j=1}^{r_f} \gamma^{(\ell)}_{f,j} v_{t,f,j}. \tag{11}$$

We apply the same native rotary phase to all sources associated with f. Complex linear decoding within each frequency, without cross-frequency mixing or an output bias, then satisfies

$$\mathbf{R}_t \mathbf{D}^{(\ell),R} = \mathbf{D}^{(\ell),R} \mathbf{Q}_t. \tag{12}$$

This permits us to rotate the shared latent before caching while retaining layer-specific decoding.

Model-specific layouts and scoring. For DeepSeek-V3.2, we use one complex source per frequency, giving $r_R = d_R = 64$, with native NeoX pairs $(0, 32), \ldots, (31, 63)$ and $\mathbf{Q}_t = \mathbf{R}_t$. For GLM-5, we use two sources per frequency, giving $r_R = 128$ and $d_R = 64$. Each 64-dimensional source block uses native interleaved real/imaginary pairs, the original position angles, and rotary base $\theta = 10^6$. In rank-major order,

$$\mathbf{Q}_t = \mathrm{diag}(\mathbf{R}_t, \mathbf{R}_t), \qquad \mathbf{D}^{(\ell),R} \in \mathbb{R}^{64\times 128}. \tag{13}$$

At inference, we absorb the decoders into post-RoPE, pre-Hadamard queries, matching the coordinate system of the reconstructed keys. We then quantize and score directly in latent coordinates, without applying Hadamard rotation or incorporating it into the decoders.

A.3 Joint NoPE Regression

For a group of m layers and n calibration tokens, let $\mathbf{X} \in \mathbb{R}^{n\times d}$ contain anchor inputs as rows, where $d = d_{\mathrm{model}}$, and concatenate the NoPE targets as $\mathbf{Y}_N = [\mathbf{K}_{\ell_1,N}, \ldots, \mathbf{K}_{\ell_m,N}] \in \mathbb{R}^{n\times md_N}$. We solve

$$\min_{\mathrm{rank}(\mathbf{B}_N)\le r_N,\, \mathbf{b}_N} \frac{1}{n}\|\mathbf{Y}_N - \mathbf{X}\mathbf{B}_N - \mathbf{1}\mathbf{b}_N^\top\|_F^2 + \lambda_N \|\mathbf{B}_N\|_F^2. \tag{14}$$

We penalize the effective mapping $\mathbf{B}_N$ and leave the bias unregularized, summing reconstruction errors across layers without energy normalization. Both models use $d_N = 64$ and $r_N = 128$.

Let $\boldsymbol{\mu}_x$ and $\boldsymbol{\mu}_y$ denote the training means, and define

$$\begin{aligned} \mathbf{C}_{xx} &= \mathbf{X}^\top \mathbf{X}/n - \boldsymbol{\mu}_x \boldsymbol{\mu}_x^\top, \\ \mathbf{C}_{xy,N} &= \mathbf{X}^\top \mathbf{Y}_N/n - \boldsymbol{\mu}_x \boldsymbol{\mu}_y^\top, \\ \mathbf{A}_N &= \mathbf{C}_{xx} + \lambda_N \mathbf{I}. \end{aligned} \tag{15}$$

With $\mathbf{V}_N$ containing the leading r_N right singular vectors of $\mathbf{A}_N^{-1/2}\mathbf{C}_{xy,N}$, we obtain

$$\mathbf{P}_N = \mathbf{A}_N^{-1}\mathbf{C}_{xy,N}\mathbf{V}_N, \qquad \mathbf{F}_N = \mathbf{V}_N^\top, \qquad \mathbf{B}_N = \mathbf{P}_N\mathbf{F}_N, \tag{16}$$

and

$$\mathbf{b}_N = \boldsymbol{\mu}_y - \mathbf{B}_N^\top \boldsymbol{\mu}_x. \tag{17}$$

We split $\mathbf{F}_N$ into blocks $(\mathbf{D}^{(\ell),N})^\top$ and $\mathbf{b}_N$ into layer-specific biases. At inference, we project uncentered inputs as $\mathbf{X}\mathbf{P}_N$; the output biases incorporate the mean correction.

A.4 Frequency-Wise Complex RoPE Regression

We pack each layer's pre-RoPE key pairs into complex coordinates. For frequency f, let $\mathbf{Y}_f \in \mathbb{C}^{n\times m}$ contain the group targets. We fit a bias-free, uncentered mapping:

$$\min_{\operatorname{rank}_{\mathbb{C}}(\mathbf{B}_f)\le r_f} \frac{1}{n}\|\mathbf{Y}_f - \mathbf{X}\mathbf{B}_f\|_F^2 + \lambda_R\|\mathbf{B}_f\|_F^2. \tag{18}$$

Define $\mathbf{A}_R = \mathbf{X}^\top\mathbf{X}/n + \lambda_R\mathbf{I}$ and $\mathbf{C}_{xy,f} = \mathbf{X}^\top\mathbf{Y}_f/n$. With $\mathbf{V}_f$ containing the leading r_f right singular vectors of $\mathbf{A}_R^{-1/2}\mathbf{C}_{xy,f}$, we obtain

$$\mathbf{P}_f = \mathbf{A}_R^{-1}\mathbf{C}_{xy,f}\mathbf{V}_f, \qquad \mathbf{F}_f = \mathbf{V}_f^H, \qquad \mathbf{B}_f = \mathbf{P}_f\mathbf{F}_f. \tag{19}$$

The columns of $\mathbf{P}_f$ generate the shared complex sources, and the corresponding layer column of $\mathbf{F}_f$ supplies the decoder coefficients $\gamma_{f,j}^{(\ell)}$. We fit frequencies independently, using $r_f = 1$ for DeepSeek-V3.2 and $r_f = 2$ for the final GLM-5 model.

A.5 Regularization Selection and Numerical Implementation

Validation criterion. We search $\mathcal{A} = \{0, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ and set

$$\lambda_c = \alpha_c \operatorname{tr}(\mathbf{C}_{xx})/d, \qquad c \in \{N, R\}. \tag{20}$$

Both branches use the anchor inputs' centered training variance to scale regularization, and all rotary frequencies within a group share one λ_R.

For each branch, we evaluate training-fitted parameters using the mean layer-wise normalized reconstruction error:

$$J_c(\alpha) = \frac{1}{m}\sum_{\ell\in\mathcal{G}_g} \frac{\sum_{t\in\text{val}} \|\widehat{\mathbf{k}}_{\ell,t,c}(\alpha) - \mathbf{k}_{\ell,t,c}\|_2^2}{\sum_{t\in\text{val}} \|\mathbf{k}_{\ell,t,c}\|_2^2}. \tag{21}$$

We accumulate errors and target energies over all validation tokens and retain the training-fitted NoPE bias. We select the strongest regularization within 0.5% of the minimum validation error:

$$\alpha_c^\star = \max\left\{\alpha\in\mathcal{A} : J_c(\alpha) \le 1.005 \min_{\alpha'\in\mathcal{A}} J_c(\alpha')\right\}. \tag{22}$$

Reported configurations. For DeepSeek-V3.2, we select regularization independently for each group and branch. All 15 groups in the 50-request calibration, $G = 4$ configuration select $\alpha_N = \alpha_R = 10^{-2}$. For $G = 8$, we refit the groups using the same calibration and selection protocol.

For GLM-5, we first calibrate a configuration with one complex source per rotary frequency. We retain its NoPE parameters and refit the rotary projections and decoders with two sources per frequency, keeping each group's previously selected rotary regularization strength fixed.

Numerical implementation. We accumulate training and validation sufficient statistics separately in FP64, using token counts, feature sums, and the products $\mathbf{X}^\top\mathbf{X}$, $\mathbf{X}^\top\mathbf{Y}$, and $\mathbf{Y}^\top\mathbf{Y}$. We assemble complex rotary statistics from real-coordinate statistics and solve in complex128. This preserves all calibration observations without storing token-level features. We evaluate inverse operations spectrally, using a tolerance-based pseudoinverse at zero regularization and zero-padding rank-deficient factors to the prescribed dimensions. We check statistics-based validation errors against explicit predictions and verify exported FP32 parameters against the high-precision solution.

B Storage and Inference Implementation

B.1 Cache Storage and Fixed Parameters

Let N_L denote the number of indexed layers, N_{ind} the number of independent native layers, M the number of sharing groups, L the history length, and b the bytes per stored element. We compute cache payloads, excluding quantization metadata and allocation overhead, as

$$\mathcal{M}_{\text{DSA}} = N_L L d_I b, \qquad \mathcal{M}_{\text{LatentIndex}} = Lb(N_{\text{ind}} d_I + M d_C). \tag{23}$$

For a group of m layers, we replace m native keys with one shared latent, giving a payload ratio of $d_C/(m d_I)$.

In our DeepSeek-V3.2 FP8 implementation, we store each native key in 132 bytes: 128 FP8 values and one FP32 scale. Each shared latent occupies 200 bytes: 192 FP8 values and separate FP32 scales for the RoPE and NoPE segments. Including the independent native layer, we obtain

$$\begin{aligned} m_{\text{DSA}} &= 61 \times 132 = 8052 \text{ bytes/token}, \\ m_{G=4} &= 132 + 15 \times 200 = 3132 \text{ bytes/token}, \\ m_{G=8} &= 132 + 8 \times 200 = 1732 \text{ bytes/token}. \end{aligned} \tag{24}$$

We thus reduce logical indexer-cache storage by 61.1% for $G = 4$. This accounting excludes allocation overhead, temporary workspaces, HS candidate indices, fixed parameters, and main-attention KV caches.

We use $d_{\text{model}} d_C$ shared projection parameters per group. Each group member additionally uses $d_N r_N$ NoPE decoder parameters, d_N bias parameters, and $2\sum_f r_f$ real rotary decoder parameters. These parameters are independent of context length.

B.2 Query Absorption and FP8 Execution

We generate the training-free latent through a linear projection without bias, activation, or additional normalization, using fitted targets that already include native key normalization. We apply RoPE to the rotary branch and cache the latent once per group. Each layer retains its native query and head-weight computations.

We perform decoder absorption on post-RoPE, pre-Hadamard queries. For NoPE, we compute $(\mathbf{D}^{(\ell),N})^\top \mathbf{q}_N$ and the bias contribution $\beta = \mathbf{q}_N^\top \mathbf{b}^{(\ell)}$. For rotary frequency f and latent source j, we compute

$$\widetilde{q}_{\ell,f,j} = \overline{\gamma^{(\ell)}_{f,j}} q_{\ell,f}. \tag{25}$$

We then score the rotated latent directly, retain the native indexer scaling, and add β before ReLU and head aggregation. We neither apply Hadamard rotation on this path nor incorporate it into the decoders.

For DeepSeek-V3.2, we use a RoPE64/NoPE128 layout. We quantize cached latents and absorbed queries separately by segment using E4M3FN, with independent scales rounded up to powers of two; cached key scales are stored in FP32. The kernel rescales the segment dot products, sums them, adds the bias contribution, applies ReLU, and aggregates heads. We export calibration parameters in FP32, perform online latent projection in BF16, and compute query absorption in FP32 before conversion to BF16 and FP8 quantization.

For GLM-5, we use a RoPE128/NoPE128 segmented FP8 layout. The rotary segment comprises two consecutive 64-dimensional source blocks with native interleaved real/imaginary pairs. We apply the same native cosine and sine values to both blocks and quantize them together as one rotary segment.

B.3 Computation With and Without Hierarchical Selection

Without HS, we independently score the full causal history and select top-k tokens at every layer. NoPE query absorption costs $O(H_I d_N r_N)$ per query, and rotary absorption costs $O(H_I r_R)$. Scoring costs $O(H_I L d_C)$ per layer per query and $O(H_I L^2 d_C)$ over prefill. Each layer separately reads the shared cache and scores in the d_C-dimensional latent space.

Table 5: Training-free model and latent configurations. Layer indices are zero-based.

Setting	DeepSeek-V3.2	GLM-5
Indexed layers N_L	61	78
Hidden dimension d_{model}	7168	6144
Indexer heads H_I	64	32
Native RoPE / NoPE (d_R, d_N)	64 / 64	64 / 64
Shared RoPE / NoPE (r_R, r_N)	64 / 128	128 / 128
Independent layer indices	0	0, 1
Grouped layer indices	1–60	2–77
Number of four-layer groups	15	19
Complex sources per rotary frequency	1	2

With HS, we let the anchor score the full prefix and each follower score at most B candidates. For a group of m layers, we therefore score

$$L + (m-1)\min(B, L) \tag{26}$$

positions per query instead of mL. For DeepSeek-V3.2 with one independent layer, 15 anchors, 45 followers, and $B = 8192$, this gives $16L + 45 \times 8192$ positions when $L > 8192$, compared with $61L$ for native DSA.

These counts cover scoring positions only. In Appendix C.4, we measure complete indexer forward calls, including candidate construction and TopK, while excluding main attention and other model components.

C Training-Free Experimental Details

C.1 Models and Comparison Settings

We compare native DSA, IndexCache, and LatentIndex on DeepSeek-V3.2 and GLM-5. Native DSA selects tokens independently at each indexed layer. For IndexCache, we reuse anchor selections with the same group boundaries as LatentIndex. All methods select $K = \min(2048, N)$ tokens for a valid causal prefix of length N.

We use $G = 4$, 50-request calibration, and segmented FP8 inference for the main results (Table 5). We keep fitted parameters fixed throughout evaluation and use the GLM-5 calibration procedure described in Appendix A.5.

C.2 Downstream Evaluation and Recall Protocol

Downstream evaluation. We evaluate RULER on all 13 tasks with 100 examples per task at each of six context lengths: 4K, 8K, 16K, 32K, 64K, and 128K. We average task scores within each length and then equally across lengths. For LongBench, we evaluate six categories: single-document QA, multi-document QA, summarization, few-shot learning, synthetic retrieval, and code completion. We use completion mode for code tasks and chat mode otherwise, averaging task scores within each category and then equally across categories. We run each method end to end using its own token selections.

Recall sampling and aggregation. At each nominal context length, we use 100 examples from each of the 13 RULER tasks. We evaluate the query at the first token of the question text, using only the causal prefix through that position; N denotes its actual length. For each example, we run the native DSA trajectory and compare all selectors at the same query using the same dense core-attention probabilities, computed over the full causal prefix with the native attention scale.

We compute head-wise oracle-normalized ratios as defined in Section 4.2, average over core-attention heads, and then over all indexed layers, including independent layers: 61 for DeepSeek-V3.2 and 78 for GLM-5. We subsequently average examples within each task and give tasks equal weight. Length averages give the six lengths equal weight. For DeepSeek-V3.2, we use all 128 core-attention heads.

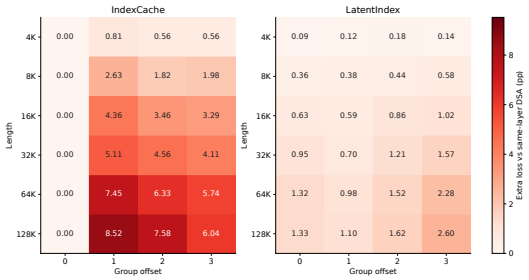


Figure 4: Additional R_{mean} loss relative to native DSA by group offset on DeepSeek-V3.2 ($G = 4$). We use the same trajectory, sampling, and aggregation as in Figure 2a.

C.3 Cross-Layer Sharing Analysis

Group offsets and group size. For the DeepSeek-V3.2 $G = 4$ analysis, we measure additional recall loss relative to each layer's native DSA selection on the same reference trajectory. We average layers at the same offset across the 15 groups, with offset 0 denoting anchors and offsets 1–3 denoting followers, then aggregate examples and tasks as above. We include independent layer 0 in overall recall but exclude it from offset analysis.

For $G = 8$, we retain layer 0 independently, form seven eight-layer groups from layers 1–56, and place layers 57–60 in one four-layer group. We retain the RoPE64/NoPE128 latent layout and refit projections and decoders using the same calibration protocol. We use identical group boundaries for IndexCache.

Mean-attention recall. Using the same dense probabilities as R_{head}, we first average probabilities across heads as $\bar{p}_i = H^{-1} \sum_h p_{h,i}$ and define the oracle $T = \text{TopK}_i(\bar{p}_i, K)$. We compute

$$R_{\text{mean}} = \frac{\sum_{i \in S} \bar{p}_i}{\max\left(\sum_{i \in T} \bar{p}_i, \epsilon\right)}, \qquad \epsilon = 10^{-8}. \tag{27}$$

This metric measures how well a token set shared across attention heads preserves head-averaged attention mass relative to the optimal shared size-K set. We use it to assess whether the advantage observed under head-wise normalization also holds for aggregate attention mass. We compute both metrics in the same evaluation pass and apply the same layer, example, and task aggregation. Figure 4 reports additional R_{mean} loss relative to native DSA using the same offset analysis.

C.4 Hierarchical Selection and Decode Indexer Timing

Candidate construction. We use $B = 8192$ candidates and final selection budget $k = 2048$ at all lengths. Layer 0 retains native DSA. In each four-layer group, the anchor scores the full prefix and constructs its candidate set from its top-2048 positions and the next 6144 highest-scoring positions. Each follower independently rescores the candidates and selects its own top-2048. We cap both budgets by the valid prefix length and rebuild candidates for each group at every decode step. At 4K, followers score the full valid prefix; at nominal 8K, measured decode prefixes exceed 8192 tokens and use the fixed candidate budget.

Workload and aggregation. We measure batch-size-one decoding on eight NVIDIA B300 GPUs with tensor parallelism 8, using nested prefixes of the same book. Each request generates 65 tokens. We exclude the first token produced by prefill and measure the following 64 decode steps, whose history lengths are approximately $L + 1$ through $L + 64$ for nominal input length L. We measure three requests per method and length.

We place CUDA events around each layer's complete `indexer.forward`. Let $t_{\ell,r}^{(j,s)}$ denote elapsed time for layer ℓ, TP rank r, request j, and decode step s. We report

$$\overline{T}_{\text{indexer}} = \frac{1}{3} \sum_{j=1}^{3} \frac{1}{64} \sum_{s=1}^{64} \sum_{\ell=0}^{60} \max_{r=0,\ldots,7} t_{\ell,r}^{(j,s)} \tag{28}$$

Table 6: Training settings. All methods completed both schedules. Batch sizes count sequences; the global indexer-loss coefficient is distinct from the method-specific layer weights in Appendix D.2.

Setting	Dense warmup	Sparse training
Sequence length	65,536	65,536
Completed optimizer steps	400	2,426
Global batch size	8	256
Microbatch size	1	1
Gradient accumulation steps	8	4
Learning rate	10^{-3}	4×10^{-5}
Learning-rate schedule	Constant	Constant
Learning-rate warmup steps	0	0
Indexer objective	Full-prefix KL	Selected-support KL
Global indexer-loss coefficient	1.0	1.0
Backbone updates	Disabled	Enabled
Precision	BF16	BF16
GPU count	8	64
TP / PP / CP / EP	1 / 1 / 8 / 8	1 / 1 / 1 / 8

in milliseconds per decode token.

D Training-Aware Experimental Details

D.1 Training Configuration

We compare DSA, IndexCache, and LatentIndex on DeepSeek-V2-Lite using eight indexer heads and top-2048 selection. For grouped methods, we partition the 27 layers into nine groups of three. Dense warmup freezes the backbone and trains only the indexers. Sparse training loads each method's own warmup checkpoint, activates its routes, and updates the backbone with next-token LM loss while continuing indexer supervision. We initialize new indexer linear weights from a zero-mean normal distribution with standard deviation 0.02, and LayerNorm weights and biases to one and zero, respectively. For sparse training, we use the same dataset, cached sample and shuffle indices, seed, and sequential sampler across methods. We use a global batch size of 256, microbatch size 1, data parallelism 64, and context parallelism 1, aligning sample slots at matched training steps.

D.2 Indexer Construction and Training Objectives

Shared latent indexers. For each group, we project the anchor's normalized hidden state into a 128-dimensional NoPE latent and a 64-dimensional RoPE key. We apply LayerNorm only to the NoPE latent and use layer-specific linear decoders to produce 64-dimensional NoPE keys. Each layer directly reuses the shared RoPE key, corresponding to an identity rotary decoder.

We generate all three layers' queries and head weights from the anchor hidden state using separate layer-specific projections. All linear projections are bias-free. After applying RoPE, we apply the same normalized Hadamard rotation to queries and keys. During sparse training, we compute three independent top-2048 selections at the group entrance for the corresponding layers to consume. During warmup, we evaluate each layer's indexer output against its own dense-attention teacher using the cached anchor input.

Attention supervision. For layer ℓ and query t, we form the teacher distribution $P_t^{(\ell)}$ by averaging the 16 core-attention heads' probabilities, using the runtime attention scale, including YaRN scaling. We form the indexer distribution $S_t^{(\ell)}$ by applying softmax to its scores. The indexer's head weights include the scaling factor $1/(\sqrt{8}\sqrt{128})$.

During warmup, both distributions cover the full causal prefix. During sparse training, we use the current layer's sparse-attention probabilities as the teacher and normalize both distributions over the

selected positions. With prediction mask m_t, we define

$$\overline{\mathcal{L}}_\ell = \frac{\sum_t m_t D_{\mathrm{KL}}\left(\mathrm{sg}[P_t^{(\ell)}] \,\|\, S_t^{(\ell)}\right)}{\sum_t m_t}, \tag{29}$$

where sg denotes stop-gradient. We detach indexer inputs from the backbone and optimize the backbone through next-token language modeling during sparse training.

Layer-wise loss normalization. All methods receive supervision from all 27 layers. DSA and LatentIndex use each layer's corresponding indexer output, while IndexCache supervises the shared anchor indexer separately with each group member's teacher. In both stages, we use

$$\mathcal{L}_{\mathrm{idx}} = \sum_{\ell=0}^{26} c_\ell \overline{\mathcal{L}}_\ell, \qquad c_\ell = \begin{cases} 1/3, & \text{IndexCache,} \\ 1, & \text{DSA and LatentIndex.} \end{cases} \tag{30}$$

We apply the training framework's microbatch and data-parallel reductions without an additional division by the number of layers or groups.

D.3 Recall Diagnostics and Plotted Training Curves

Warmup Recall@2048. For each sampled query s and layer ℓ, we measure selected attention mass relative to each head's top-K oracle, where $K = \min(2048, N)$ for causal prefix length N:

$$r_{\ell,s} = \frac{1}{16} \sum_{h=1}^{16} \frac{\sum_{t \in \mathcal{I}_{\ell,s}} \widetilde{p}_{s,t,h}^{(\ell)}}{\sum_{t \in \mathcal{O}_{\ell,s,h}} \widetilde{p}_{s,t,h}^{(\ell)}}. \tag{31}$$

Here $\mathcal{I}_{\ell,s}$ is the diagnostic indexer selection, $\mathcal{O}_{\ell,s,h}$ is the head-specific oracle, and $\widetilde{p}$ is the diagnostic dense-attention distribution.

We sample at most two evenly spaced local queries per layer and context-parallel rank, using CP=8. We average query-level recall within each layer, then across ranks and all 27 layers. All methods use the same sampling procedure and data order. For IndexCache followers, we score with the anchor's queries, keys, and head weights against each follower's own dense-attention reference.

We compute diagnostic attention probabilities with scale $1/\sqrt{576}$, corresponding to the absorbed MLA query width $512 + 64$. The diagnostic therefore uses the head-wise oracle normalization of R_{head}, with its own attention scaling and query sampling.

LM loss and plotting. We report next-token cross-entropy over valid prediction tokens, with EOD loss masking and excluding auxiliary losses. We normalize within each microbatch by its valid-token count, then average across microbatches and data-parallel ranks. Each logged value corresponds to its optimizer step. We compute method-minus-DSA differences at matched steps without interpolation and apply bias-corrected causal EMA after differencing, using coefficients 0.9 for warmup and 0.6 for sparse training.