FLASHINDEXER: DYNAMIC HEAD-ROUTED INDEXER FOR TOKEN-LEVEL SPARSE ATTENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Token-level sparse attention, exemplified by DeepSeek Sparse Attention (DSA), significantly reduces the downstream attention computation, but shifts the computational burden to the lightweight indexer. For every query, the indexer must compute scores for all context tokens using its entire set of heads before selecting the most relevant top-k tokens, resulting in a per-layer cost that grows with both the context length and the number of indexer heads. Recent work has reduced indexer computation along the token and layer axes, but the head axis has received comparatively little attention. We revisit the head axis and make two key observations. First, the contributions of indexer heads are highly imbalanced and vary substantially across samples. Second, the important heads for each query can be reliably identified using only the query itself, without accessing any context tokens. Motivated by these observations, we propose **FlashIndexer**, a plug-and-play head-routing module for the DSA indexer. For each query, it dynamically selects a small subset of important heads from all indexer heads, activating only the selected heads for scoring context tokens, and thereby substantially reducing indexer computation. This query-adaptive routing preserves the original token-level top-k selection while avoiding unnecessary computation from less informative heads. With only a small subset of indexer heads activated, FlashIndexer matches the accuracy of the original full-head DSA indexer on DeepSeek-V4-Flash-0731 and GLM-5.1 across RULER and LongBench v2, while achieving up to $1.20\times$ end-to-end time-to-first-token speedup at 128K context.

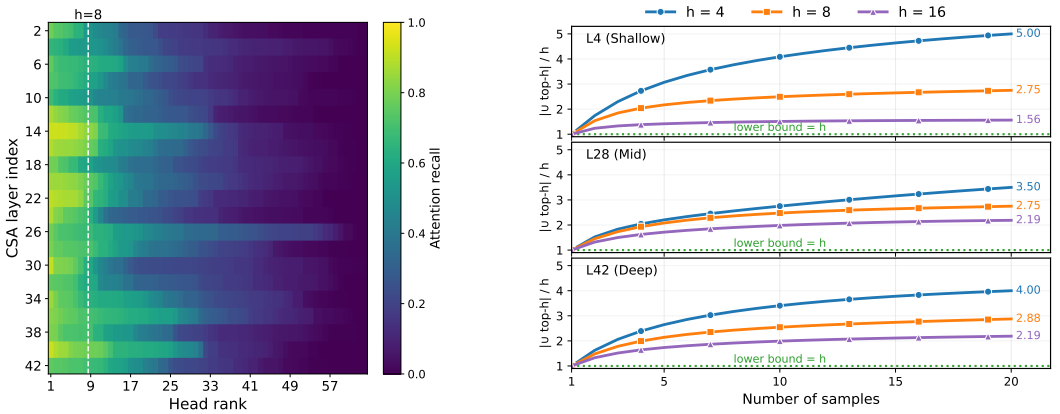
1 INTRODUCTION

The growing demand for long-context inference has made attention efficiency a central challenge for large language models (LLMs). DeepSeek Sparse Attention (DSA) (Liu et al., 2025) addresses this with fine-grained token-level sparse attention: a lightweight indexer identifies the context tokens most relevant to each query, and only the selected tokens participate in the subsequent sparse attention computation. While this substantially reduces the cost of the subsequent attention computation, the indexer must still score every context token using its entire set of heads to determine the token-level top-k set. Consequently, the indexing cost scales with both the context length and the number of indexer heads, making the indexer a new computational bottleneck. This cost is particularly pronounced during prefill, where every query position is scored against its entire preceding context, resulting in a per-layer indexing cost that grows quadratically with the prompt length.

Recent work reduces this overhead by exploiting redundancy along distinct dimensions of the indexing process. Along the token axis, hierarchical indexing (Xu et al., 2026b) reduces the number of context tokens that require fine-grained evaluation from the key side, while query-group methods (Liu et al., 2026) amortize token selection across neighboring positions from the query side. Along the layer axis, index reuse methods (Bai et al., 2026) share token selections across Transformer layers. These results reveal substantial redundancy in indexing, yet leave the head axis largely unexplored: whichever tokens or layers are selected, token relevance is still scored by every indexer head. This motivates us to examine complementary redundancy along the head axis.

To characterize this redundancy, we analyze how individual indexer heads contribute to token selection. As shown in Figure 1, head contributions are markedly imbalanced: a small subset of heads accounts for a large fraction of the attention recall, while the remaining heads contribute much less.

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.



(a) Indexer head contributions are imbalanced

(b) Top-h head union increases with samples

Figure 1: Head importance in the DSA indexer. (a) The attention recall heatmap ranks heads independently within each layer, revealing highly imbalanced head contributions, with a small number of top-ranked heads accounting for a large fraction of the attention recall. (b) Across shallow, middle, and deep layers, the union of top-h heads grows with the number of samples, indicating that important indexer heads vary across samples.

More importantly, the top-h heads vary substantially across samples: as more samples are considered, the union of their top-h heads continues to grow across different layers, indicating that different samples rely on different subsets of indexer heads. Thus, redundancy is inherently *input-dependent*: the important head subset can vary across samples and token positions. Statically pruning a fixed subset of heads therefore cannot fully exploit this redundancy; the indexer should instead allocate computation dynamically to the heads that matter for each token.

This raises a central design question: ***how can important heads be identified without introducing significant additional overhead?*** Existing approaches such as MISA (Zhou et al., 2026) estimate head importance using a lightweight module, but still require accessing prefix-side tokens, introducing additional context-dependent computation. Meanwhile, recent studies (Zhang et al., 2022; Jin et al., 2024; Piękos et al., 2025) show that lightweight routers can make input-dependent selection decisions directly from the current token representation, without requiring access to the full context. This suggests that important indexer heads can likewise be selected from the current token.

Building on these observations, we propose **FlashIndexer**, a dynamic head-routed indexer that identifies important heads directly from the hidden state of the current token, without accessing the context, and activates only a small subset for token scoring. The selected heads follow the original DSA indexing procedure, preserving the existing token selection and downstream sparse attention computation. Specifically, we freeze the original DSA indexer and optimize only the lightweight router. We further develop a token-packing kernel that combines the active heads of multiple query tokens into larger matrix operations, thereby realizing the computational savings from head reduction. Moreover, FlashIndexer is orthogonal to existing token- and layer-axis indexing optimizations.

Our contributions are summarized as follows:

- We characterize the head-axis redundancy in the DSA indexer and show that important heads are highly input-dependent, yet can be reliably identified directly from the current token representation alone, without accessing the context.
- We propose FlashIndexer, a plug-and-play head-routing approach that activates only a small subset of indexer heads while preserving the original DSA token-level top-k interface. The router is trained independently on top of the frozen DSA indexer.
- We co-design a token-packing GPU kernel that combines the active heads of multiple query tokens into larger matrix operations, so that the reduced head count translates into wall-clock gains. On DeepSeek-V4-Flash-0731 and GLM-5.1, activating h=8 heads matches the full-head accuracy on RULER and LongBench v2, while delivering up to $1.20\times$ end-to-end time-to-first-token speedup at 128K context. Moreover, FlashIndexer further composes with IndexCache, achieving up to $1.75\times$ speedup.

2 Related Work

Long-context and sparse attention. To reduce the quadratic cost of long-context attention, early approaches impose fixed sparsity patterns, such as local and strided attention (Child et al., 2019; Beltagy et al., 2020). Later methods introduce input-dependent sparsity at the block level, dynamically identifying relevant KV blocks for each query (Gao et al., 2024; Jiang et al., 2024; Tang et al., 2026; Qiu et al., 2026; Niu et al., 2026; Xue et al., 2026; Lu et al., 2025; Yuan et al., 2025; Tang et al., 2024). These methods exploit coarse-grained sparsity to reduce attention computation while retaining efficient block-wise execution on modern hardware. In contrast, token-level methods select individual context tokens, providing fine-grained and more content-dependent sparsity. DeepSeek Sparse Attention (DSA) (Liu et al., 2025) adopts this paradigm with a learned indexer that scores context tokens for each query and selects a token-level top-k set for subsequent sparse attention. Recent work has further demonstrated the scalability of token-level sparse attention in large language models (Zeng et al., 2026; Xu et al., 2026a; Zan et al., 2026), making fine-grained token selection an increasingly important approach for long-context inference.

Reducing DSA indexer cost along different axes. Several studies have explored reducing the computational cost of DSA-style indexing along different axes. HISA (Xu et al., 2026b) adopts hierarchical indexing to reduce the number of prefix tokens requiring fine-grained scoring along the token axis. PIVOT (Liu et al., 2026) reduces redundant prefix scans by amortizing indexing across groups of neighboring queries, targeting the query axis. IndexCache (Bai et al., 2026) exploits cross-layer redundancy by reusing selected indices across Transformer layers. These approaches reduce the amount of indexing computation by limiting the tokens or layers on which the indexer is invoked, while the invoked indexer still evaluates token relevance using all of its heads. More recently, MISA (Zhou et al., 2026) explores sparsity along the head axis by selecting a subset of indexer heads for token scoring. It estimates head importance from prefix-side pooled statistics, coupling head selection with the cached prefix and requiring pooled key representations to be maintained alongside the cache. In contrast, FlashIndexer performs head selection from the query side using a learned gate, without accessing prefix keys during routing. This difference allows the routing overhead to remain independent of context length and leads to a different implementation.

Mixture-of-head and routed attention. Conditional computation over attention heads has also been explored beyond DSA indexers. MoA (Zhang et al., 2022) selects a subset of attention heads for each token, MoH (Jin et al., 2024) formulates multi-head attention as a mixture-of-head module with token-dependent routing, and MoSA (Piękos et al., 2025) investigates content-dependent routing for sparse attention. These approaches apply conditional routing to the attention computation itself, with the selected heads contributing directly to the attention output. In contrast, FlashIndexer applies head routing to the auxiliary indexer: the selected heads only determine which prefix tokens are retained for the subsequent sparse attention, while the original attention operator remains unchanged.

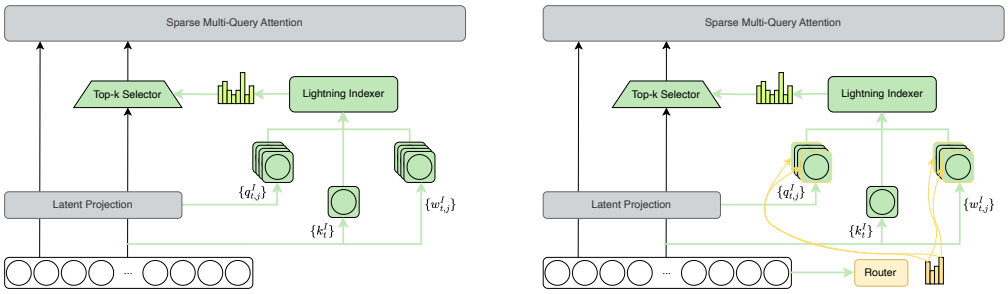
3 Method

We first briefly review the DeepSeek Sparse Attention (DSA) (Liu et al., 2025) indexer and establish the notation used throughout this section. For a token at position t with a causal prefix of length L, the indexer uses H_I indexer heads. Each head $j \in \{1, \dots, H_I\}$ is associated with an indexing query $q^I_{t,j}$ and a per-token gating weight $w^I_{t,j}$, while each prefix position s has an indexing key k^I_s that incorporates rotary position information (Su et al., 2024). The relevance score for prefix position s and the resulting token selection are defined as

$$I_{t,s} \;=\; \sum_{j=1}^{H_I} w^I_{t,j} \cdot \mathrm{ReLU}\big(q^I_{t,j} \cdot k^I_s\big), \qquad \mathcal{T}_t \;=\; \mathrm{TopK}\big(I_{t,:},\, k\big), \tag{1}$$

where $I_{t,:} = (I_{t,1}, \dots, I_{t,L})$ is the score vector over the causal prefix, and $\mathrm{TopK}(\cdot, k)$ returns the indices of its k largest entries. Computing the indexer scores costs $O(H_I L)$ per token and $O(H_I L^2)$ per layer during prefill.

The key observation underlying FlashIndexer is that head importance is input-dependent: different query tokens rely on different subsets of indexer heads, while each token can be served by only a



(a) Original DSA indexer (b) FlashIndexer

Figure 2: Comparison of the original DSA indexer and FlashIndexer. (a) The original indexer sends all H_I indexer heads to the prefix-wide scoring path before token-level top-k selection. (b) FlashIndexer introduces a per-token router to choose $h \ll H_I$ active heads; only the selected heads participate in prefix-token scoring. Both paths produce the same token-level top-k interface for the downstream sparse multi-query attention operator.

small subset of them. FlashIndexer (Figure 2) turns this observation into a mixture-of-head indexer with four components. First, a lightweight router conditioned on the current hidden state selects $h \ll H_I$ active heads for each query token (Section 3.1). Second, only the selected heads scan the causal prefix to compute the token-level index scores (Section 3.2). Third, we train only the router with a selected-head objective while freezing the original backbone and indexer parameters (Section 3.3). Finally, an efficient kernel computes the prefix scores for only the activated heads, translating the reduced head computation into lower prefill latency (Section 3.4).

3.1 Per-Token Head Routing

For each query token at position t, FlashIndexer maps the current layer hidden state $x_t \in \mathbb{R}^d$ to one routing score per indexer head:

$$g_t^I \;=\; W_g^I x_t \in \mathbb{R}^{H_I}, \qquad W_g^I \in \mathbb{R}^{H_I \times d}, \tag{2}$$

where d is the model hidden size and W_g^I is a learnable router projection. We define the active-head set as the h heads with the largest routing scores:

$$\mathcal{H}_t \;=\; \mathrm{TopK}\big(g_t^I,\, h\big), \qquad h \ll H_I, \tag{3}$$

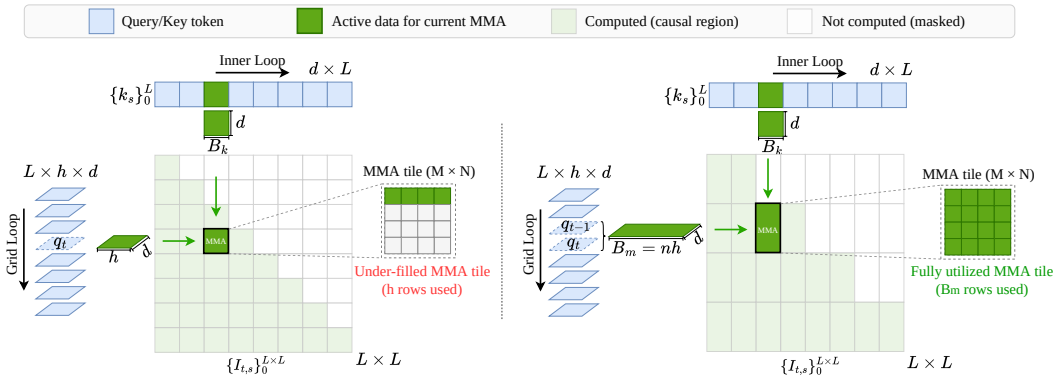
where $\mathrm{TopK}(\cdot, h)$ returns the indices of the h largest entries. For each query token t, $\mathcal{H}_t$ is selected only once, and the resulting h active heads are used to score every token in its causal prefix. This design reduces computation along the head axis while preserving token-level scoring over the full prefix. Computing $\mathcal{H}_t$ requires only one $d \times H_I$ projection and a top-h selection per token, with an overhead independent of the prefix length L.

3.2 Sparse Token Scoring

Given the active set $\mathcal{H}_t$, FlashIndexer scores the full context tokens with only the selected heads:

$$\widehat{I}_{t,s} \;=\; \sum_{j \in \mathcal{H}_t} g_{t,j}^I \, w_{t,j}^I \cdot \mathrm{ReLU}\big(q_{t,j}^I \cdot k_s^I\big), \qquad \widehat{\mathcal{T}}_t \;=\; \mathrm{TopK}\big(\widehat{I}_{t,:},\, k\big), \tag{4}$$

where $\widehat{I}_{t,:} = (\widehat{I}_{t,1}, \ldots, \widehat{I}_{t,L})$ collects the routed scores over the context tokens, and $\widehat{\mathcal{T}}_t$ denotes the selected top-k token set. For each active head, the routing score $g_{t,j}^I$ serves two roles: it ranks the heads for selection and modulates the contribution of each selected head by scaling its original DSA gating weight $w_{t,j}^I$. Compared with the full-head formulation in Equation 1, the head aggregation is therefore reduced from H_I heads to h heads. The prefix-scoring cost decreases from $O(H_I L)$ to $O(hL)$ per token; including the router projection, the total cost is $O(hL + dH_I)$. Because reducing the number of heads does not change the indexer's output interface, FlashIndexer can be directly plugged into the DSA pipeline, balancing inference efficiency and token-selection quality.



(a) Per-token execution (original kernel)

(b) Token-packed execution (optimized kernel)

Figure 3: Kernel optimization for FlashIndexer. (a) Per-token execution under-utilizes the MMA tile when only h heads are active. (b) Token packing combines multiple queries into $B_m = nh$ rows, improving MMA utilization and reducing the number of grid-loop iterations. This enables the reduction in active heads to translate into effective kernel-level speedup.

3.3 Training Objective

We retain the KL-divergence objective used in the sparse training stage of DeepSeek-V3.2 (Liu et al., 2025). For each query token t, the target distribution $p_{t,\mathcal{T}_t}$ is derived from the downstream attention over the original DSA-selected token set $\mathcal{T}_t$, while the student distribution is obtained by normalizing the routed indexer scores over the same token set. The training objective is

$$\mathcal{L}^I = \mathbb{E}_t \left[\mathrm{KL}\left(p_{t,\mathcal{T}_t} \,\middle\|\, \mathrm{Softmax}\left(\widehat{I}_{t,\mathcal{T}_t} \right) \right) \right], \tag{5}$$

which trains the routed indexer to reproduce the token-selection behavior captured by the original DSA indexer. Our training procedure differs from the original in two aspects.

Only the router is trained. Rather than updating the indexer as in DeepSeek-V3.2, we freeze all original parameters, and update only the newly introduced router parameters W_g^I. The original indexer is therefore kept fixed, and training only learns how to route among its heads.

Active-head scoring with teacher-forced token selection. The student distribution is computed from the routed scores $\widehat{I}_{t,s}$ in Equation 4, where only the activated heads $\mathcal{H}_t$ contribute to each score. Importantly, the distillation target $p_{t,\mathcal{T}_t}$ is constructed using the original DSA token set $\mathcal{T}_t$, rather than the routed set $\widehat{\mathcal{T}}_t$. This teacher-forced construction keeps the selected tokens consistent with the original DSA indexer during training, preventing inaccurate top-k predictions from introducing layer-wise error accumulation in the early stages of training.

3.4 Kernel Design

Reducing the number of active heads lowers the head-token scoring cost, but does not directly translate into proportional wall-clock speedup. As shown in Figure 3, the original score execution processes each query token independently, so a small number of active heads leaves the matrix multiply-accumulate (MMA) unit under-utilized. We instead pack multiple query tokens from the same request, combining their active heads into a larger MMA operation. This improves MMA utilization and reduces the number of grid-loop iterations required to process the queries. FlashIndexer adopts this token-packing strategy to turn the reduced head count into effective kernel-level speedup.

This optimization primarily benefits the prefill stage, where multiple query tokens are available for packing and can be processed together. During decode, each request typically contains only a single query token, leaving insufficient active heads to fully utilize an MMA operation. The resulting under-utilization limits the practical benefit of head sparsity. FlashIndexer therefore applies the optimized execution to prefill while leaving the decode path unchanged.

4 EXPERIMENT

4.1 EXPERIMENTAL SETUP

Models. We evaluate FlashIndexer on two token-level sparse attention models built on the DSA indexer: **DeepSeek-V4-Flash-0731** (DSV4-Flash) (Xu et al., 2026a) (H_I=64 indexer heads, token top-k=512) and **GLM-5.1** (GLM-5.1-Team, 2026) (H_I=32 indexer heads, token top-k=2048). Unless stated otherwise, FlashIndexer activates h=8 indexer heads per query and applies head routing only during the *prefill* stage, while retaining the original indexer computation during decode.

Router training. FlashIndexer keeps all original model parameters frozen and updates only the router (Section 3.3). We adapt the router on long-context corpora—an in-house 128K-token code corpus for DSV4-Flash and the public GLM-5.1-Reasoning-1M-Cleaned for GLM-5.1—which is lightweight, with 1B training tokens sufficient for both models. Corpus and full training hyperparameters are given in Appendix A.2.

Benchmarks. We evaluate on **RULER** (Hsieh et al., 2024) at 32K, 64K, and 128K (13 tasks, 1300 examples per length), reporting the per-length mean and its average across lengths (AVG), and on **LongBench v2** (Bai et al., 2025) at 128K (260 examples, up to five runs), reporting the official aggregate (Overall) and its six task domains (SingleQA, MultiQA, ICL, Dialog, Code, Struct); per-domain sizes and task details are in Appendix A.3. Efficiency is evaluated separately in Section 4.3, where we report both kernel-level performance and end-to-end prefill latency and speedup.

Baselines. We compare **FlashIndexer** against representative indexer-acceleration methods spanning different forms of redundancy: the full-head DSA indexer (**DSA**); **HISA** (Xu et al., 2026b), token-axis hierarchical block indexing (block size 128, top-64 blocks); **MISA** (Zhou et al., 2026), head-axis mixture-of-experts (single-stage v1, 1024-token pooling, 8 heads/query); and **IndexCache** (Bai et al., 2026), layer-axis cross-layer index reuse (index computed on $6/21$ layers for DSV4-Flash and $22/78$ for GLM-5.1, reused on the rest; per-layer patterns in Appendix A.3). Following the evaluation protocol above, all methods are applied only during prefill, with the original indexer computation retained during decode. Since FlashIndexer is orthogonal to token- and layer-axis acceleration, we additionally combine it with IndexCache, denoted as **FlashIndexer-IC**.

4.2 MAIN RESULTS: ACCURACY

Table 1 reports accuracy for both models on RULER across context lengths and on LongBench v2 by task domain. Across both DSV4-Flash and GLM-5.1, FlashIndexer closely matches the full-head DSA baseline while activating only a small fraction of indexer heads. In contrast, HISA degrades noticeably as the context length increases, dropping to 83.97 at 128K on DSV4-Flash and 80.96 on GLM-5.1. Its block-level top-m filtering commits to coarse token blocks and becomes increasingly prone to discarding information required by later queries, suggesting that coarse-grained token-axis selection is less robust at long contexts. MISA, which also exploits head-axis sparsity, achieves competitive accuracy overall, further supporting the presence of substantial redundancy across indexer heads. Nevertheless, a clear gap remains on GLM-5.1 at 128K RULER, where MISA reaches 93.05 compared with 94.16 for FlashIndexer and 94.04 for full-head DSA. Finally, because FlashIndexer operates purely along the head axis, it naturally composes with layer-axis reuse. We therefore also report its combination with IndexCache (FlashIndexer-IC), which preserves accuracy up to a small change while providing additional prefill-latency gains, as discussed in Section 4.3.

4.3 EFFICIENCY

We measure end-to-end efficiency by time-to-first-token (TTFT) on a random-input workload (batch size 1, single decoded token, mean over five repeats). At these context lengths TTFT is dominated by prefill and thus reflects FlashIndexer's prefill savings; Figure 4(a) reports the speedup over full-head DSA. It grows with input length—routing overhead is query-side and context-independent while the eliminated indexing cost scales with the prefix—reaching 1.20× on GLM-5.1 at 128K. MISA instead falls below 1× on short inputs (0.84× at 8K), paying its prefix-dependent routing cost before

Table 1: Accuracy on RULER and LongBench v2. "Heads" is the number of indexer heads activated per token. FlashIndexer (h=8) matches full-head DSA while activating only a small fraction of indexer heads. AVG is the mean over the three RULER lengths and Overall is the official aggregate LongBench v2 score; benchmark settings are described in Section 4.1. Best results are in bold and second-best are underlined, ranked among the indexer-acceleration methods.

		RULER				LongBench v2						
Method	Heads	32K	64K	128K	AVG	SingleQA	MultiQA	ICL	Dialog	Code	Struct	Overall
DSV4-Flash (H_I=64, top-k=512)												
DSA	64	96.09	96.16	93.59	95.28	64.4	61.2	71.0	84.0	74.5	45.0	65.85
HISA	64	92.62	88.38	83.97	88.32	61.8	60.5	71.6	84.0	80.0	30.0	64.54
MISA	8	**96.08**	95.69	93.82	95.20	64.6	**60.7**	69.0	84.0	81.8	**45.0**	65.85
IndexCache	64	95.53	94.40	92.90	94.28	**66.5**	60.2	68.4	83.0	78.2	**45.0**	**66.23**
FlashIndexer	8	**96.08**	**95.75**	**93.85**	**95.23**	65.8	**60.7**	67.1	83.0	80.0	40.0	65.92
FlashIndexer-IC	8	95.31	94.30	92.94	94.18	64.1	58.3	**72.3**	**85.0**	**85.5**	**45.0**	65.46
GLM-5.1 (H_I=32, top-k=2048)												
DSA	32	95.62	94.02	94.04	94.56	63.7	64.2	69.0	85.0	80.0	50.0	66.62
HISA	32	93.79	85.98	80.96	86.91	64.1	63.2	**71.6**	83.0	**81.8**	**50.0**	66.69
MISA	8	**95.61**	**92.99**	93.05	93.88	**64.6**	63.2	69.7	84.0	78.2	**50.0**	66.62
IndexCache	32	94.29	92.20	92.37	92.95	64.3	**63.8**	71.0	83.3	78.8	**50.0**	**66.79**
FlashIndexer	8	95.44	**92.99**	**94.16**	**94.20**	**64.6**	62.2	**71.6**	85.0	74.5	**50.0**	66.46
FlashIndexer-IC	8	93.99	92.24	91.31	92.51	64.2	61.0	**71.6**	**87.0**	72.7	**50.0**	66.00

any savings accrue. Being orthogonal to layer-axis reuse, FlashIndexer composes with IndexCache to reach $1.75\times$ at 128K.

Figure 4(b) attributes these gains at the kernel level, decomposing the per-layer indexer latency (QK projection, head router, indexer logits, top-k) normalized to DSA. The logits module—all H_I heads over the full prefix—dominates the DSA cost and shrinks to less than half once only h=8 heads are active. The methods diverge on the router: MISA's prefix-dependent routing grows with context and offsets part of its savings, whereas FlashIndexer's per-token router stays nearly flat, yielding up to $1.52\times$ per-layer speedup at 128K on GLM-5.1 (against $1.30\times$ for MISA). The shared QK projection and top-k are unaffected by head routing.

Figure 4(c) isolates the logits kernel, where head reduction translates most directly into compute savings: activating h=8 of H_I heads achieves up to a $2.78\times$ speedup at 128K on GLM-5.1. The remaining gap to the ideal $4\times$ reduction from the head-count ratio mainly comes from non-compute overheads in the kernel, whose contribution becomes increasingly amortized at longer context lengths.

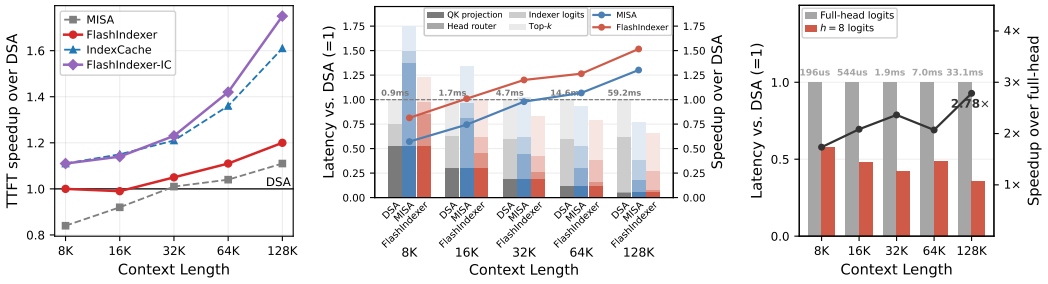
The $2.78\times$ speedup of the logits kernel translates to $1.52\times$ at the indexer-layer level, since the QK projection and top-k computation remain unchanged, and further to $1.20\times$ end-to-end, as the indexer logits account for a substantial share of the overall prefill latency. As the context length increases, the logits computation accounts for an even larger fraction of the prefill cost, leading to larger end-to-end gains. The corresponding analysis for DSV4-Flash is given in Appendix A.4 (Figure 7).

4.4 Analysis and Ablations

4.4.1 Routing Quality

The accuracy results show that head routing preserves the full-head selection; Figure 5 now examines the routing signal itself, independently of end-to-end accuracy. For each query we take the token-level top-k selection of full-head DSA, $\mathcal{T}_t$ in Equation 1, as the reference and compare it with the selection $\widehat{\mathcal{T}}_t$ produced by a routed indexer that scores the prefix using only the h heads ranked highest by a given signal (hereafter $\widehat{\mathcal{T}}_t$ always denotes the selection under the ranking being evaluated). The *attention-score recall* is the attention mass on the overlap of the two selections divided by that on the reference,

$$\mathrm{Recall}_h(t) \;=\; \frac{\sum_{s\in\mathcal{T}_t\cap\widehat{\mathcal{T}}_t} a_{t,s}}{\sum_{s\in\mathcal{T}_t} a_{t,s}}, \qquad |\mathcal{T}_t| = k, \tag{6}$$



(a) End-to-end TTFT (b) Per-layer breakdown (c) Logits kernel

Figure 4: Efficiency of FlashIndexer on GLM-5.1 at three levels of granularity; the corresponding results for DSV4-Flash are in Appendix A.4 (Figure 7). (a) TTFT speedup over full-head DSA at input lengths from 8K to 128K. (b) Per-layer indexer latency normalized to DSA; each group of bars is one context length, bars within a group are DSA, MISA, and FlashIndexer, and darker-to-lighter shading marks the QK projection, head router, indexer logits, and top-k modules, with DSA bars annotated by absolute per-layer latency and the lines (right axis) showing the per-layer speedup. (c) Logits-kernel latency normalized to DSA with absolute full-head latencies annotated; the line (right axis) is the resulting kernel speedup.

where $a_{t,s}$ is the downstream attention score of token s. Recall measures selection overlap in attention-mass terms; as a parallel measure from the *attention-output* side that cross-validates it, we report the reconstruction error between the sparse attention computed on the routed selection and that computed on the full-head reference,

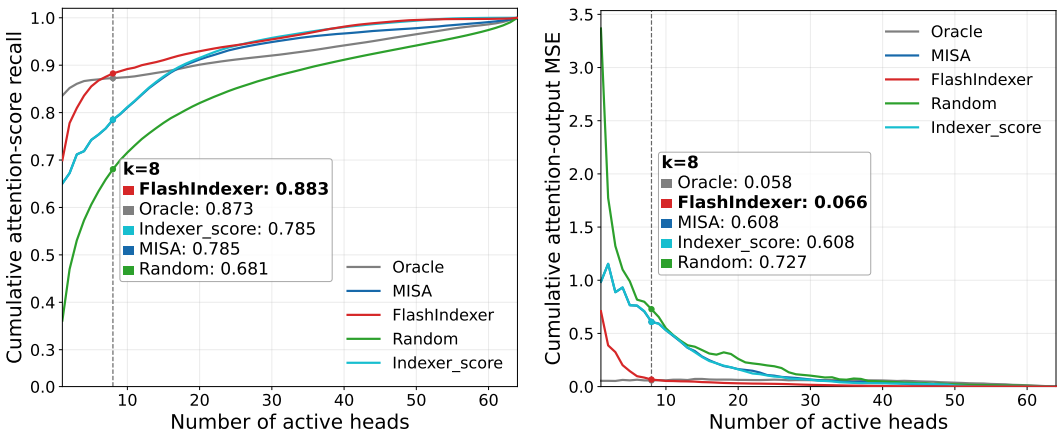
$$\mathrm{MSE}_h(t) \;=\; \frac{1}{d}\big\|\mathbf{o}_t(\widehat{\mathcal{T}}_t) - \mathbf{o}_t(\mathcal{T}_t)\big\|_2^2, \qquad \mathbf{o}_t(\mathcal{S}) = \sum_{s\in\mathcal{S}} \tilde{a}_{t,s}\,\mathbf{v}_s, \tag{7}$$

where $\mathbf{o}_t(\mathcal{S})$ is the sparse-attention output over a token set $\mathcal{S}$ with the softmax renormalized on $\mathcal{S}$, so $\mathbf{o}_t(\mathcal{T}_t)$ uses the full-head reference and $\mathbf{o}_t(\widehat{\mathcal{T}}_t)$ the routed h-head selection; lower is better, averaged over samples and layers. We compare five rankings: **FlashIndexer**'s learned gate, **MISA**'s prefix-side importance, the **indexer score**, which ranks heads by the mean absolute per-head score over the prefix, $\mathrm{mean}_s\big|w^I_{t,j}\,\mathrm{ReLU}(q^I_{t,j}\cdot k^I_s)\big|$, a **Random** ranking, and an **Oracle** ranking heads per query by each head's individual value on the metric being plotted—recall in Figure 5(a) and reconstruction MSE in Figure 5(b). The indexer score is MISA's scoring rule at pooling block size one—the no-pooling limit computed over every prefix key rather than 1024-token blocks—and thus upper-bounds what prefix-dependent routing of this form can reach, at the cost of a full prefix pass.

In Figure 5(a) all curves rise steeply at small h, confirming that a small fraction of indexer heads recovers most of the full-head selection; the comparison then isolates each signal. Random ordering is strong but clearly suboptimal, showing the head pool is redundant in aggregate yet that *which* heads are chosen still matters. The two prefix-dependent signals coincide at h=8 (0.785 for both MISA and the exact indexer score), so dropping the pooling approximation does not help: MISA's 1024-token blocks already extract all the routing information the signal carries, and the gap is a property of the signal, not the approximation. FlashIndexer's learned gate exceeds this ceiling, reaching 0.883 against 0.681 for random, and tracks the per-query oracle across the whole range of h—even edging past it (0.873), since the recall-maximizing subset need not coincide with the individually strongest heads. Two conclusions follow: head utility is not predicted by score magnitude, and the query alone identifies near-optimal head subsets, so the prefix pass that prefix-dependent routing pays for buys no additional routing quality—consistent with both the negligible accuracy loss in Table 1 and the latency advantage in Section 4.3. The attention-output MSE in Figure 5(b) confirms this from the output side: the ordering is unchanged, and at h=8 FlashIndexer attains the lowest reconstruction error among the practical signals, essentially matching the oracle.

4.4.2 Ablation: Number of Active Heads h

We vary the number of active indexer heads, $h \in \{1, 4, 8, 16\}$, to characterize the accuracy–efficiency trade-off on LongBench v2 (Table 2), using the same 5-run protocol as Table 1. Accuracy



(a) Attention-score recall (b) Attention-output MSE

Figure 5: Routing-signal quality as a function of the number of active indexer heads, on DSV4-Flash (H_I=64; 20 RULER samples at 16K, averaged over samples, layers, and 1K tokens sampled at random per sample). Each curve ranks heads by one of the five signals defined in the text; the Oracle ranks heads per token by their individual recall in (a) and individual reconstruction MSE in (b). **(a)** Attention-score recall (Eq. equation 6); higher is better. **(b)** Attention-output MSE (Eq. equation 7); lower is better.

Table 2: Effect of the number of active heads h on DSV4-Flash. Accuracy is reported on LongBench v2 (128K, 5-run averages) with the same task-domain breakdown as Table 1; h=64 corresponds to DSA (full head).

h	SingleQA	MultiQA	ICL	Dialog	Code	Struct	Overall
1	63.2	60.5	65.8	84.0	76.4	35.0	64.38
4	63.7	61.2	66.5	84.0	78.2	45.0	65.15
8	65.8	60.7	67.1	83.0	80.0	40.0	**65.92**
16	65.3	60.2	69.0	85.0	80.0	30.0	65.77
64 (full)	64.4	61.2	71.0	84.0	74.5	45.0	65.85

degrades noticeably at h=1 (64.38 Overall), recovers to 65.15 at h=4, and reaches 65.92 at h=8, already matching full-head DSA (65.85). Performance then saturates, with h=16 achieving a comparable score of 65.77. These results show that a small routed subset of heads is sufficient to recover the full-head selection signal. We therefore adopt h=8 as the smallest setting that matches DSA across metrics, while keeping top-k and decoding fixed so that the sweep isolates the effect of h.

5 Conclusion

We have presented **FlashIndexer**, a dynamic head-routed indexer for token-level sparse attention. Motivated by the observation that indexer head contributions are both highly imbalanced and input-dependent, FlashIndexer uses a lightweight per-token router to activate only a small subset of indexer heads for each query and scores the prefix with those heads alone. Because routing depends only on the current token, its overhead is independent of context length. As a plug-and-play module, FlashIndexer preserves the original DSA token-level top-k interface and trains only a small router on top of a frozen model. On DSV4-Flash and GLM-5.1, activating just 8 indexer heads matches full-head accuracy on RULER and LongBench v2, composes with existing token- and layer-axis optimizations, and—together with a co-designed token-packing kernel—turns head sparsity into practical prefill-stage speedups. More broadly, our results show that the head axis is a complementary and largely untapped source of redundancy in sparse-attention indexers, and that combining head routing with token- and layer-axis methods is a promising path toward more efficient long-context inference.

AI USE STATEMENT

In this work, we did not use generative AI tools for any task with required disclosure: we have not used them to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate or prove mathematical claims, propose or refine hypotheses, design or provide feedback on our research methodology or experiments, implement methods, assist with translation, clean or reformat datasets, or interpret results. Additionally, we used generative AI tools for tasks with recommended disclosure: drafting and editing the paper to improve its readability, suggesting a structure for the paper, and creating and editing the Python code used to produce the figures in Section 4. We have reviewed and verified all AI-assisted text and code, and the authors checked every generated figure for correctness. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

FlashIndexer is an inference-efficiency method: a lightweight, plug-and-play router trained on top of a frozen sparse-attention model. Our study does not involve human subjects, personally identifiable information, or the collection of any new data. Evaluation uses established public benchmarks (RULER and LongBench v2), and router adaptation uses a publicly available chain-of-thought corpus (GLM-5.1-Reasoning-1M-Cleaned) together with an in-house code corpus, each used in accordance with its license and intended research use. Because FlashIndexer only reduces the number of indexer heads used for token selection and leaves the frozen backbone and downstream sparse-attention interface unchanged, it does not introduce new model capabilities and neither amplifies nor mitigates the biases or potential harms of the underlying pretrained models; it inherits whatever properties those models already have. The primary societal effect we anticipate is positive: by lowering the indexing cost of long-context inference, FlashIndexer reduces the compute and energy required to serve large language models. We declare no conflicts of interest, and we are not aware of any ethical concerns raised by this work.

REPRODUCIBILITY STATEMENT

We have taken several steps to make our results reproducible. The complete FlashIndexer formulation—per-token head routing, sparse token scoring, the training objective, and the token-packing kernel—is specified with full notation and equations in Section 3 (Eqs. equation 2–equation 5). The experimental protocol, including the two backbone models and their indexer configurations, the evaluation benchmarks and their settings, and the baselines, is described in Section 4.1, and the efficiency measurement procedure (TTFT workload, per-layer breakdown, and kernel-level timing) is detailed in Section 4.3. Appendix A.2 reports the full router training recipe—training corpora, all optimizer hyperparameters, sequence lengths, token budgets, hardware and parallelism configuration, and training curves (Table 3, Figure 6)—and emphasizes that all pretrained parameters are frozen so that only the router projection W_g^I is learned. Per-task RULER and per-domain LongBench v2 results, the routing-quality protocol, and the ablations that support our claims are given in Section 4.4 and Appendices A.3, A.5, and A.6. The GLM-5.1-Reasoning-1M-Cleaned corpus is publicly available at the URL cited in Appendix A.2, and both backbone models—DeepSeek-V4-Flash-0731 and GLM-5.1—are publicly released open-source models. The only component we cannot release is the in-house 128K-token code corpus used to adapt the router on DeepSeek-V4-Flash-0731; the router recipe itself is fully specified above and applies to any long-context corpus, and the GLM-5.1 experiments rely entirely on publicly available models and data, so they can be reproduced end to end from the details above.

REFERENCES

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. 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.

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.

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

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, et al. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

GLM-5.1-Team. Glm-5.1: Towards long-horizon tasks. `https://z.ai/blog/glm-5.1`, 2026. Accessed: 2026-04-07.

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? *arXiv preprint arXiv:2404.06654*, 2024.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. Moh: Multi-head attention as mixture-of-head attention. *arXiv preprint arXiv:2410.11842*, 2024.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

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, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

Lin Niu, Xin Luo, Linchuan Xie, Yifu Sun, Guanghua Yu, Jianchen Zhu, and S Kevin Zhou. Stem: Rethinking causal information flow in sparse attention. *arXiv preprint arXiv:2603.06274*, 2026.

Piotr Piękos, Róbert Csordás, and Jürgen Schmidhuber. Mixture of sparse attention: Content-based learnable sparse attention via expert-choice routing. *arXiv preprint arXiv:2505.00315*, 2025.

Quantong Qiu, Zhiyi Hong, Yi Yang, Haitian Wang, Kebin Liu, Qingqing Dang, Juntao Li, and Min Zhang. Flux attention: Context-aware hybrid attention for efficient llms inference. *arXiv preprint arXiv:2604.07394*, 2026.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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

Zecheng Tang, Quantong Qiu, Yi Yang, Zhiyi Hong, Haiya Xiang, Kebin Liu, Qingqing Dang, Juntao Li, and Min Zhang. Elastic attention: Test-time adaptive sparsity ratios for efficient transformers. *arXiv preprint arXiv:2601.17367*, 2026.

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

Yufei Xue, Lin Niu, Hong Liu, Siran Liu, Hanyong Shao, Wei Liu, Guanghua Yu, Jianchen Zhu, and Jun Zhang. Cosa: Accelerating long-context inference via proxy-kernel co-designed sparse attention. *arXiv preprint arXiv:2607.25291*, 2026.

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, et al. Longcat sparse attention: Taming the lightning via streaming-aware hierarchical cross-layer indexing. *arXiv preprint arXiv:2608.01662*, 2026.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. Mixture of attention heads: Selecting attention heads per token. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 4150–4162, 2022.

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.

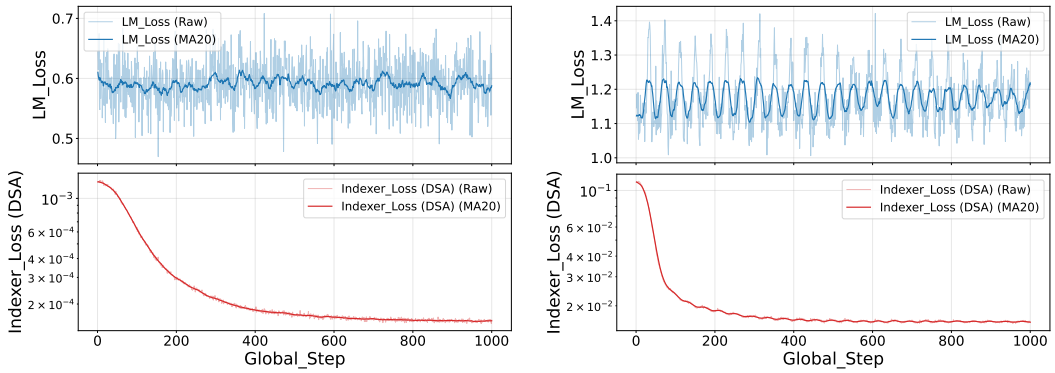
A Appendix

A.1 Limitations

Decode-stage routing. FlashIndexer applies head routing only during prefill and retains the full-head indexer during decode. The token-packing kernel that turns the reduced head count into wall-clock savings (Section 3.4) combines the active heads of many query tokens into larger matrix operations, which is only possible when many queries are processed together, as in prefill; a decode step scores a single query token against the KV cache and cannot be packed the same way. Extending per-token head routing to decode—for instance by packing across the batch dimension or across speculative-decoding drafts—is a natural next step.

Post-hoc router adaptation. We freeze a fully trained model and adapt only the router (Section 3.3), so the router can exploit only the head structure that pretraining already produced and cannot reshape which heads specialize. Training the router jointly with the indexer during pretraining could instead let the model learn head assignments that are easier to predict from the query alone, potentially improving both routing quality and the accuracy–efficiency trade-off. We do not explore this here because it requires access to the pretraining pipeline.

Fixed head budget. FlashIndexer activates a fixed number of heads h for every query, layer, and token. Our analysis (Figure 1) shows that head redundancy is input-dependent and varies with depth, so a fixed budget may over-provision easy queries and under-provision hard ones. An adaptive budget that spends more heads only where the routing signal is uncertain could improve the trade-off further.



(a) DSV4-Flash (b) GLM-5.1

Figure 6: Router training curves. For each model the upper panel shows the language-modeling loss and the lower panel the indexer loss of Eq. equation 5 on a logarithmic axis; light lines are per-step values and solid lines a 20-step moving average.

A.2 Router Training Details

Corpora. For DSV4-Flash we adapt the router on an in-house 128K-token code corpus, with sequences truncated to 64K tokens during training to match the long-context regime in which the indexer operates. For GLM-5.1 we use GLM-5.1-Reasoning-1M-Cleaned,[1] a publicly available chain-of-thought corpus of 746K English and Chinese examples spanning general reasoning and instruction following, graduate-level science, multilingual STEM, and mathematics.

Hyperparameters. Table 3 lists the router training hyperparameters for both models. As in Section 3.3, only the router projection W_g^I receives gradients, making this a short adaptation run rather than a full fine-tune: 1000 steps over 64K-token sequences, roughly 1B tokens per model. Both runs share the same optimizer schedule (cosine decay from $1e{-}4$ to $1e{-}5$ with 50 warmup steps, AdamW with $\beta = (0.9, 0.95)$, weight decay 0.1, gradient clipping 1.0), weight the indexer loss of Eq. equation 5 by 0.01 against the language-modeling objective, and run in bf16 with master gradients.

The two runs differ only in the hardware and parallelism needed to fit each model. DSV4-Flash uses 16 GPUs across 4 nodes (pipeline parallelism 2, context parallelism 8) and GLM-5.1 uses 32 GPUs across 4 nodes (pipeline parallelism 2, context parallelism 16). Both shard MoE experts—over 8 and 16 expert-parallel groups respectively—on the same devices rather than additional ones, so the pipeline and context degrees alone set the device count, and each optimizer step consumes 16 sequences through gradient accumulation.

Training curves. Figure 6 shows the router adaptation on both models. The indexer loss of Eq. equation 5—the only source of gradients, since every other parameter is frozen—drops by roughly an order of magnitude: from $1.057e{-}3$ to $1.254e{-}4$ on DSV4-Flash ($8.4\times$) and from 0.1060 to 0.01529 on GLM-5.1 ($6.9\times$) over the full 1000 steps, confirming that the router learns to reproduce the full-head target distribution within the 1B-token budget. The language-modeling loss is reported for reference only: with the backbone frozen it tracks the training data rather than the router. Neither run shows loss spikes or divergence, supporting the claim that adapting the router is a lightweight and stable procedure.

A.3 Experiment and Data Details

Table 1 reports the RULER task-mean score, which aggregates 13 tasks of 100 examples each. Tables 4 and 5 break that mean down per task for DSV4-Flash and GLM-5.1 respectively, following the method ordering of Table 1; the AVG column reproduces the corresponding RULER entry there.

[1] https://huggingface.co/datasets/Jackrong/GLM-5.1-Reasoning-1M-Cleaned

Table 3: Router training hyperparameters for DSV4-Flash and GLM-5.1.

	DSV4-Flash	GLM-5.1
Indexer heads H_I / active h	64 / 8	32 / 8
Sequence length	64K	64K
Trained parameters	router only	router only
GPUs (nodes × GPUs)	4×4	4×8
Pipeline / context par.	2 / 8	2 / 16
Expert par. (shares devices)	8	16
Recomputation	full, uniform	full, uniform
Steps	1000	1000
Tokens	∼1B	∼1B
Sequences per step	16	16
Optimizer	AdamW	AdamW
Learning rate	1e−4	1e−4
Min. learning rate	1e−5	1e−5
Schedule / warmup	cosine / 50	cosine / 50
(β_1, β_2)	$(0.9, 0.95)$	$(0.9, 0.95)$
Weight decay / grad. clip	0.1 / 1.0	0.1 / 1.0
Indexer loss weight	0.01	0.01
Precision	bf16 (O2)	bf16 (O2)

We abbreviate the tasks as S1–S3 (single-needle NIAH), MK1–MK3 (multi-key NIAH), MV (multi-value NIAH), MQ (multi-query NIAH), VT (variable tracking), CWE (common-word extraction), FWE (frequent-word extraction), and QA1/QA2 (SQuAD and HotpotQA).

The breakdown shows that most retrieval tasks are saturated: S1–S3, MQ, MV, and VT stay at or near 100 for every method and context length, so the differences in task-mean score are driven by a small number of tasks. This is where the token-axis and head-axis methods separate most clearly. HISA's degradation is concentrated almost entirely in the multi-key tasks—on DSV4-Flash at 128K it falls to 55.0 on MK2 and 29.0 on MK3 against 100.0 for full-head DSA, and GLM-5.1 shows the same pattern (49.0 and 34.0)—consistent with block-level token pruning discarding keys that are individually needle-like but not salient at block granularity. FlashIndexer, by contrast, tracks DSA within a point on every task, including the multi-key tasks that HISA loses (98.0/99.0/100.0 on MK1–MK3 at 128K). The remaining non-saturated tasks are FWE and the two QA tasks, on which all methods—including full-head DSA—degrade as the context grows; FlashIndexer stays within the run-to-run spread of DSA there as well.

LongBench v2 domains. The 260 LongBench v2 examples span the six task domains reported in Table 1: Single-Document QA (SingleQA, 113), Multi-Document QA (MultiQA, 81), Long In-context Learning (ICL, 31), Long-dialogue History Understanding (Dialog, 20), Code Repository Understanding (Code, 11), and Long Structured Data Understanding (Struct, 4); results are averaged over up to five runs.

Head-redundancy analysis. The analysis of Figure 1 is measured on DeepSeek-V4-Flash-0731. Panel (a) is a single representative sample from RULER at 64K, whose per-layer attention-recall heatmap illustrates the imbalanced head contributions. Panel (b) uses 20 samples drawn from four 64K-token sources—RULER, LongBench v2, and two in-house corpora (code and chat)—so that the growth of the top-h head union across samples reflects input variation across task types rather than the idiosyncrasies of any single benchmark.

IndexCache layer pattern. IndexCache marks each indexer-bearing layer as either *full* (`F`, which computes the token-level top-k selection) or *shared* (`S`, which reuses the most recent full layer's

Table 4: Per-task RULER scores on DSV4-Flash (H_I=64, top-k=512). Task abbreviations are defined in Section A.3; AVG is the task mean reported in Table 1.

Method	S1	S2	S3	MK1	MK2	MK3	MV	MQ	VT	CWE	FWE	QA1	QA2	AVG
32K context														
DSA	100.0	100.0	100.0	100.0	100.0	100.0	99.2	100.0	98.8	100.0	100.0	78.1	73.0	96.09
HISA	100.0	100.0	99.0	99.0	84.0	70.0	97.8	99.8	99.8	100.0	100.0	80.8	74.0	92.62
MISA	100.0	99.0	100.0	100.0	100.0	100.0	99.2	100.0	100.0	100.0	99.7	78.1	73.0	**96.08**
IndexCache	100.0	100.0	100.0	99.0	100.0	100.0	98.8	100.0	100.0	100.0	99.0	78.2	67.0	95.53
FlashIndexer	100.0	100.0	99.0	100.0	100.0	100.0	99.5	100.0	100.0	99.0	100.0	79.5	72.0	**96.08**
FlashIndexer-IC	100.0	99.0	100.0	100.0	100.0	98.0	99.0	100.0	100.0	99.9	100.0	76.2	67.0	95.31
64K context														
DSA	100.0	99.0	100.0	100.0	100.0	100.0	99.0	100.0	99.6	100.0	98.7	78.8	75.0	96.16
HISA	100.0	98.0	100.0	96.0	74.0	52.0	94.0	97.5	100.0	100.0	98.3	71.2	68.0	88.38
MISA	100.0	100.0	100.0	100.0	98.0	100.0	99.0	99.8	99.8	99.9	96.0	76.5	75.0	95.69
IndexCache	100.0	99.0	99.0	100.0	100.0	96.0	99.8	99.8	100.0	98.9	96.0	68.8	70.0	94.40
FlashIndexer	100.0	99.0	99.0	99.0	100.0	100.0	98.2	100.0	100.0	99.0	97.0	79.5	74.0	**95.75**
FlashIndexer-IC	100.0	99.0	99.0	100.0	100.0	99.0	99.2	99.0	99.8	99.3	93.7	68.8	69.0	94.30
128K context														
DSA	100.0	97.0	100.0	100.0	100.0	100.0	98.0	100.0	99.6	99.3	80.7	75.1	67.0	93.59
HISA	100.0	98.0	100.0	91.0	55.0	29.0	90.2	95.8	100.0	97.9	93.7	71.1	70.0	83.97
MISA	100.0	99.0	99.0	100.0	96.0	98.0	97.5	100.0	99.6	99.0	85.0	78.5	68.0	93.82
IndexCache	100.0	99.0	100.0	100.0	98.0	100.0	98.2	100.0	100.0	94.4	78.3	72.8	67.0	92.90
FlashIndexer	100.0	99.0	100.0	98.0	99.0	100.0	97.8	99.8	99.8	98.9	86.0	72.8	69.0	**93.85**
FlashIndexer-IC	100.0	99.0	100.0	99.0	96.0	99.0	99.2	100.0	100.0	93.8	79.7	73.5	69.0	92.94

selection). Read from the first to the last indexer layer, the per-layer patterns are

DSV4-Flash (6 full of 21 indexer layers):

`FSSSSSFSFSSFSSSSSFSFS`

GLM-5.1 (22 full of 78 layers):

`FFSFSSSFSSFSFSSSSSSSFSSSSSSFSSSFSFSSSSFFFFFSSSFFSSSFSSSSSSSSFSSS`
`FSSSSSSFSFSSSS`

FlashIndexer-IC composes FlashIndexer's head routing with the same layer pattern.

A.4 Efficiency Results on DSV4-Flash

Figure 7 presents the three efficiency views for DSV4-Flash, mirroring Figure 4 in the main text. The qualitative trends are identical to GLM-5.1; we report the model-specific numbers here.

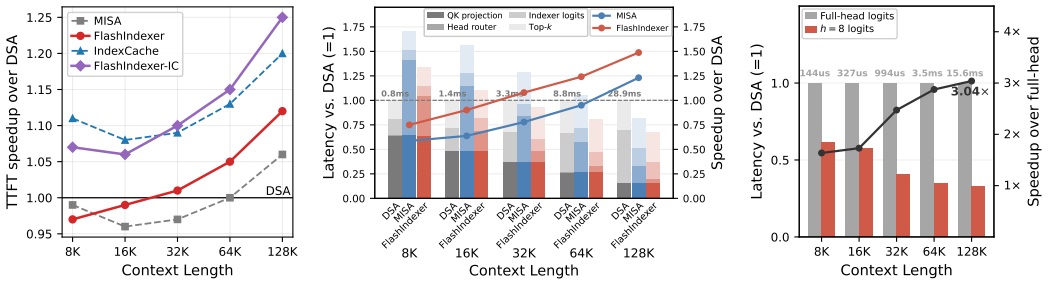
End-to-end TTFT (a). FlashIndexer's speedup over full-head DSA grows with the input length and reaches $1.12\times$ at 128K, and composing with layer-axis reuse lifts FlashIndexer-IC to $1.25\times$. As on GLM-5.1, MISA pays its prefix-dependent routing cost upfront and only turns faster at long inputs.

Per-layer breakdown (b). The logits module again dominates the DSA cost and shrinks once only h=8 of H_I=64 heads are activated, while FlashIndexer's query-side router stays nearly flat with context, giving up to $1.49\times$ per-layer speedup at 128K against $1.23\times$ for MISA. The absolute indexing cost is lower than on GLM-5.1 because DSV4-Flash uses C4-compressed indexer keys, which leaves less room for per-layer speedup—hence $1.49\times$ here versus GLM-5.1's $1.52\times$.

Logits kernel (c). Activating h=8 of 64 heads yields up to $3.04\times$ at 128K, approaching the $8\times$ head-count ratio as the fixed key-reading cost amortizes over the longer prefix. These gains compose as Amdahl's law predicts: the $3.04\times$ logits-kernel gain dilutes to $1.49\times$ per indexer layer, since the QK projection and top-k are unchanged, and to $1.12\times$ end-to-end, since the indexer is only a fraction of the full prefill pass.

Table 5: Per-task RULER scores on GLM-5.1 (H_I=32, top-k=2048). Task abbreviations are defined in Section A.3; AVG is the task mean reported in Table 1.

Method	S1	S2	S3	MK1	MK2	MK3	MV	MQ	VT	CWE	FWE	QA1	QA2	AVG
32K context														
DSA	100.0	100.0	100.0	100.0	100.0	100.0	99.5	100.0	100.0	97.0	82.7	84.8	79.0	95.62
HISA	100.0	100.0	100.0	100.0	92.0	92.0	97.8	100.0	100.0	100.0	73.7	84.9	79.0	93.79
MISA	100.0	100.0	100.0	100.0	100.0	100.0	98.8	100.0	100.0	98.0	83.7	84.5	78.0	**95.61**
IndexCache	100.0	100.0	100.0	100.0	100.0	100.0	99.2	100.0	100.0	100.0	63.3	85.2	78.0	94.29
FlashIndexer	100.0	100.0	100.0	100.0	100.0	100.0	99.2	100.0	100.0	98.0	80.0	85.5	78.0	95.44
FlashIndexer-IC	100.0	100.0	100.0	100.0	100.0	100.0	99.7	100.0	100.0	99.0	60.3	84.8	78.0	93.99
64K context														
DSA	100.0	100.0	100.0	100.0	100.0	100.0	99.0	100.0	100.0	97.8	64.0	84.5	77.0	94.02
HISA	100.0	99.0	100.0	95.0	67.0	64.0	91.5	99.5	100.0	97.9	40.7	83.2	80.0	85.98
MISA	100.0	99.0	100.0	100.0	100.0	99.0	98.5	100.0	100.0	96.5	55.0	84.8	76.0	**92.99**
IndexCache	100.0	100.0	100.0	100.0	100.0	100.0	98.8	100.0	100.0	94.7	43.7	82.5	79.0	92.20
FlashIndexer	100.0	100.0	100.0	100.0	99.0	100.0	98.2	100.0	100.0	96.1	53.3	84.2	78.0	**92.99**
FlashIndexer-IC	100.0	100.0	100.0	100.0	100.0	100.0	98.0	100.0	100.0	90.3	48.3	84.5	78.0	92.24
128K context														
DSA	100.0	100.0	99.0	99.0	97.0	100.0	98.0	100.0	100.0	90.7	71.3	86.5	81.0	94.04
HISA	99.0	98.0	100.0	89.0	49.0	34.0	84.8	97.0	100.0	79.5	65.3	82.8	74.0	80.96
MISA	100.0	100.0	100.0	100.0	95.0	99.0	98.8	100.0	100.0	85.3	69.3	85.2	77.0	93.05
IndexCache	100.0	100.0	100.0	100.0	98.0	100.0	97.2	100.0	100.0	73.7	71.7	82.2	78.0	92.37
FlashIndexer	100.0	100.0	99.0	99.0	98.0	99.0	98.5	100.0	100.0	84.4	84.0	83.2	79.0	**94.16**
FlashIndexer-IC	100.0	100.0	100.0	99.0	98.0	98.0	98.25	100.0	100.0	71.9	59.0	84.8	78.0	91.31



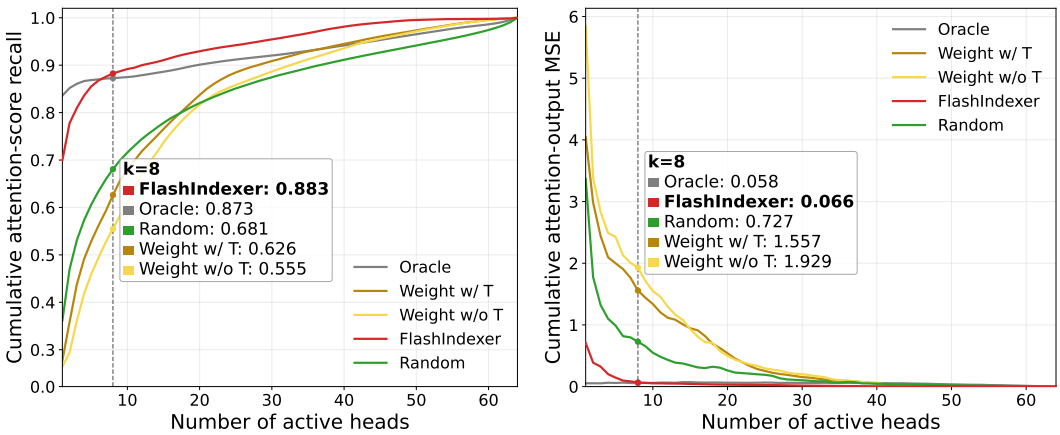
(a) End-to-end TTFT (b) Per-layer breakdown (c) Logits kernel

Figure 7: Efficiency of FlashIndexer on DSV4-Flash (C4-compressed indexer keys); panels mirror Figure 4, with all measurement settings identical to Section 4.3.

A.5 Why a Dedicated Router Is Needed

The DSA indexer already contains a per-query, per-head scalar: the gating weight $w^I_{t,j}$ of Eq. equation 1, which the indexer uses to combine the per-head scores. Since $w^I_{t,j}$ is computed from the query alone and is available before any prefix key is read, it is a natural candidate for head routing at *zero* additional cost—no new parameters and no extra projection. We explored this option before settling on the learned gate of Eq. equation 2, in two variants: taking the pretrained $w^I_{t,j}$ as the routing score directly (*Weight w/o T*), and additionally unfreezing the weight projection so that w^I can learn to serve as a router (*Weight w/ T*), with the same objective and token budget as the FlashIndexer router.

Figure 8(a) shows why we rejected both. Measured with the protocol of Section 4.4.1, neither variant reaches the random-ranking baseline: at the operating point h=8, the frozen weights recover 0.555 of the reference attention mass and the trained weights 0.626, against 0.681 for a random head subset and 0.883 for the FlashIndexer gate. Routing by the native weights is therefore not merely a weaker signal than a learned router—it is *worse than chance*, so the magnitude of $w^I_{t,j}$ is anti-correlated with the utility of head j for selecting tokens. This is consistent with its role in Eq. equation 1: the weight calibrates the *scale* at which a head's ReLU scores enter the sum, which is not the same quantity



(a) Attention-score recall (b) Attention-output MSE

Figure 8: Head rankings derived from the native indexer weights (Weight w/o T and Weight w/ T, defined in the text), measured with the protocol and probe set of Figure 5; the Oracle, Random, and FlashIndexer curves are those of Figure 5. **(a)** Attention-score recall (Eq. equation 6); higher is better. **(b)** Attention-output MSE (Eq. equation 7); lower is better.

as how informative that head is for the token-level top-k. Unfreezing the projection recovers part of the gap ($0.555 \rightarrow 0.626$) but still falls short of chance, indicating that the conflict is structural: the same projection cannot simultaneously calibrate score magnitudes for the surviving heads and rank heads for selection. The attention-output MSE in Figure 8(b) mirrors this: both native-weight rankings incur a larger reconstruction error than a random head subset at h=8, confirming that the anti-correlation between w^I magnitude and head utility carries through to the downstream attention output.

A dedicated router removes this conflict at a cost of a single $H_I \times d$ projection, which Section 3 shows to be context-independent and negligible in the latency breakdown. We therefore adopt the learned gate and leave w^I to its original role.

A.6 Why We Do Not Use Load Balancing

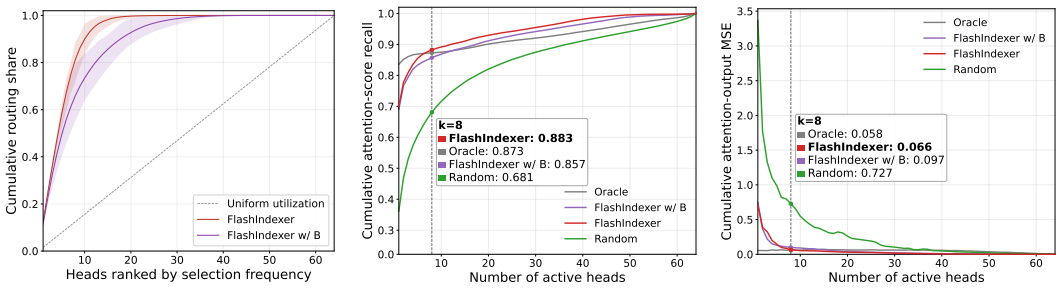
Routing a fixed number of experts per token is normally paired with a load-balancing mechanism, since an unconstrained router tends to collapse onto a few experts and leave the rest untrained (Jin et al., 2024). We implemented the auxiliary-loss-free variant of this idea (Liu et al., 2024) for the indexer: each router keeps a per-head selection counter, and after every optimizer step a non-gradient update shifts a per-head selection bias $b_{\ell,j}$ towards the heads that were selected less often than average,

$$b_{\ell,j} \leftarrow b_{\ell,j} + \eta \operatorname{sign}\big(\bar{c}_\ell - c_{\ell,j}\big), \qquad c_{\ell,j} \leftarrow 0, \tag{8}$$

where $c_{\ell,j}$ counts the tokens routed to head j of layer ℓ since the last update and $\bar{c}_\ell$ is their mean over heads. The bias enters the top-h selection only; it never scales the indexer scores, so it steers which heads are chosen without changing the routed computation.

We measure head utilization on the probe set of Section 4.4.1, accumulating the selection counts $c_{\ell,j}$ separately for each of the 21 layers that carry a Compressed Sparse Attention (CSA) indexer—the C4-compressed DSA indexer variant that DSV4-Flash uses (Section 4.1). All statistics are computed *within* a layer and only then aggregated across layers: head identity is not comparable across layers—each layer has its own router and its own head pool—and Eq. equation 8 likewise operates per layer. Pooling counts across layers before measuring concentration would inflate the apparent uniformity.

Routing is in fact strongly concentrated. Without load balancing, the normalized utilization entropy is 0.604 (mean over layers, range $[0.517, 0.664]$, where 1 denotes uniform), and on average 25 of the 64 heads in a layer are never selected at all. The load violation $\mathrm{MaxVio} = (\max_j c_{\ell,j} - \bar{c}_\ell)/\bar{c}_\ell$ reaches 6.941, essentially its ceiling of $H_I/h - 1 = 7$, which is attained only when a head is selected by every query: in most layers at least one head is active for nearly all queries.



(a) Head utilization (b) Attention-score recall (c) Attention-output MSE

Figure 9: FlashIndexer routing with and without the load-balancing update of Eq. equation 8 (DSV4-Flash, probe set of Figure 5). **(a)** Head utilization: within each layer heads are sorted by selection frequency and the curve shows the cumulative share of routing decisions captured by the top-ranked heads (solid lines are the layer mean, bands span the 10th–90th percentile); the dashed diagonal is uniform utilization, so the further a curve bows above it the more concentrated the routing. The unbalanced router is FlashIndexer, the paper's default, and the load-balanced variant is FlashIndexer w/ B. Panels (b) and (c) measure routing quality as in Figure 5; the Oracle, Random, and FlashIndexer curves are those of Figure 5. **(b)** Attention-score recall (Eq. equation 6); higher is better. **(c)** Attention-output MSE (Eq. equation 7); lower is better.

Load balancing does what it is designed to do. It raises the entropy to 0.713, cuts the number of never-selected heads roughly in half to 14.9, and relaxes the peak load in the layers where it was least extreme (MaxVio range $[3.868, 6.999]$). Figure 9(a) shows the cumulative utilization curves moving towards the uniform diagonal. This flattening, however, costs routing quality: under the protocol of Section 4.4.1, recall at $h{=}8$ drops from 0.883 to 0.868, and the balanced router stays below the unbalanced one over the entire range of h (Figure 9(b)). The attention-output side confirms this: the reconstruction MSE between the routed and full-head sparse attention (Eq. equation 7) is higher for the balanced router across the whole range of h (Figure 9(c)), so the balancing penalty shows up both in selection overlap and in the resulting attention output.

The two observations are consistent once the effect of freezing is taken into account. Load balancing exists to prevent a self-reinforcing failure in which rarely selected experts receive little gradient, fail to improve, and are therefore selected even less often. That mechanism cannot operate here: the indexer projections q^I and w^I are frozen and only the router is trained, so an unselected head is not an untrained head. It retains the scoring behavior it was pretrained with and contributes immediately whenever the router selects it. The concentration we observe is therefore a statement about head utility rather than a training pathology, and pushing utilization towards uniform spends part of the h-head budget on heads that the reference selection shows to be less useful—which is what the recall gap measures. All results reported in this paper use the unbalanced router.