SparDA: Sparse Decoupled Attention for Efficient Long-Context LLM Inference

Yaosheng Fu[1], Guangxuan Xiao[2]*, Xin Dong[3]*, Song Han[1,4], Oreste Villa[1]
[1]NVIDIA [2]Thinking Machines Lab [3]ByteDance Seed [4]MIT

Abstract

Sparse attention reduces compute and memory bandwidth for long-context LLM inference. However, two key challenges remain: (1) KV cache capacity still grows with sequence length, and offloading to CPU memory introduces a PCIe transfer bottleneck; (2) sparse selection during prefill retains $O(T^2)$ complexity and can dominate attention cost at long contexts. We propose SparDA, a decoupled sparse attention architecture that introduces a fourth per-layer projection, the *Forecast*, alongside Query, Key, and Value. The Forecast predicts the KV blocks needed by the next layer, enabling lookahead selection that overlaps CPU-to-GPU prefetch with current-layer execution. Because Forecast is decoupled from the attention query, our GQA implementation uses one Forecast head per GQA group, reducing selection overhead versus the original multi-head selector. On the evaluated 8B models, SparDA adds $<$0.5% parameters and trains only the Forecast projections using attention-derived supervision from the frozen backbone. On two sparse-pretrained 8B models, SparDA maintains accuracy comparable to the full-query sparse-attention baseline while delivering up to 1.25$\times$ prefill speedup and 1.7$\times$ decode speedup over the sparse-attention offload baseline. By enabling larger feasible batch sizes on a single GPU, SparDA further reaches up to 5.3$\times$ higher decode throughput than the non-offload sparse baseline on MiniCPM4.1-8B. Our adaptation to GLM-5.1 (754B MoE) also achieves accuracy comparable to native token-level DSA, extending the accuracy evaluation beyond block-sparse 8B models. Our source code is available at `https://github.com/NVlabs/SparDA`.

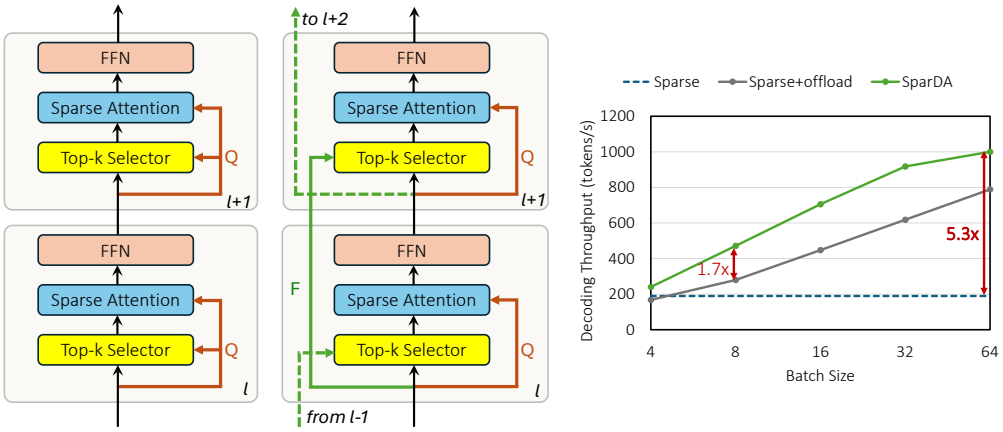
1 Introduction

Modern LLM applications increasingly require long context windows, especially for agentic and long-reasoning use cases [42, 27]. This imposes three major efficiency challenges on the underlying serving infrastructure: 1) high compute demand for attention in the prefill phase; 2) high memory bandwidth pressure for attention in the decode phase; and 3) high KV cache capacity pressure in the decode phase. Sparse attention [34, 39, 21] is a promising technique that has been adopted by recent frontier models [7, 12, 8] to improve long-context LLM inference efficiency. However, most sparse attention methods primarily reduce attention compute and memory bandwidth; while some also shrink the KV cache, capacity pressure persists as context lengths continue to grow.

A straightforward solution to this problem is to offload the KV cache to CPU memory and only fetch selected KV blocks to GPU memory for sparse attention dynamically during the decode phase. However, CPU fetching through the PCIe interface is still much slower than fetching from GPU memory, which becomes a bottleneck for long-context decoding. Prior work [41, 14] proposes both system- and algorithm-level optimizations to reduce CPU fetching traffic. InfiniGen [17] further shows that lookahead prefetching can hide part of this latency, but it relies on the raw hidden state as a proxy for future attention and can be inaccurate when adjacent-layer similarity breaks down.

*Work done while at NVIDIA.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Sparse Attention (b) SparDA (c) SparDA Speedup

Figure 1: Overview of SparDA. The Forecast $\mathbf{F}_l$ drives top-k selection for layer $l{+}1$, while $\mathbf{Q}_l$ performs sparse attention, decoupling selection from attention and exposing lookahead KV prefetch from CPU. On MiniCPM4.1-8B at 128K on H100, SparDA achieves up to 1.7× decode speedup over sparse offload and 5.3× higher throughput than non-offload sparse via larger feasible batches. The dashed line marks the non-offload sparse throughput at its largest feasible batch (B4).

Meanwhile, sparse attention itself introduces additional overhead from sparse selection. During prefill, a fixed sparse-attention budget reduces attention computation from $O(T^2)$ to $O(T)$, where T is the sequence length, but sparse selection remains $O(T^2)$ and can dominate the attention module as context grows. IndexCache [3] and HISA [32] reduce this overhead through cross-layer index reuse and hierarchical filtering, respectively.

To address the above issues, we propose SparDA, a decoupled sparse attention architecture that introduces a fourth per-layer projection, the *Forecast*, alongside the standard Query, Key, and Value (Figure 1). Rather than treating the previous hidden state as a proxy, SparDA trains this Forecast to estimate next-layer KV importance. SparDA has three key innovations:

- **Trainable lookahead sparse selection.** In standard sparse attention, the attention query drives both top-k selection and sparse attention within the same layer. SparDA decouples these roles: the Forecast generated in layer l drives top-k selection for layer $l{+}1$, while the attention query still performs sparse attention, enabling CPU-to-GPU KV cache prefetching to overlap with the current layer's execution.
- **Compact Forecast indexer.** Once Forecast is decoupled from the attention query, it no longer needs to preserve the full query-head structure for sparse selection. In our GQA implementation, the Forecast indexer uses a single Forecast head per GQA group, substantially reducing sparse-selection overhead and skipping softmax during selection at inference.
- **Asynchronous prefetch with persistent UVA kernel.** We implement the asynchronous prefetch pipeline with a persistent Unified Virtual Addressing (UVA) Triton kernel that performs high-throughput host-to-device transfers in parallel with the main GPU kernels, making sparse-attention offloading practical for large-batch long-context decoding.

SparDA can be integrated into sparse-pretrained models by training only the Forecast projections (<0.5% additional parameters on our 8B models) via Kullback–Leibler (KL) divergence against training targets derived from the frozen backbone, without retraining the base model. We evaluate SparDA on MiniCPM4.1-8B [22] and NOSA-8B [14], two sparse-pretrained 8B models, and find accuracy comparable to each model's full-query sparse-attention baseline on the evaluated long-context and long-reasoning benchmarks. In efficiency, SparDA achieves up to 1.25× prefill speedup and 1.7× decode speedup over the sparse-attention offload baseline. By enabling larger feasible batch sizes on a single GPU, SparDA reaches up to 5.3× higher decode throughput than the non-offload sparse baseline on MiniCPM4.1-8B. We further demonstrate SparDA's architectural transfer to GLM-5.1, a 754B mixture-of-experts model with native token-level DSA [35], achieving accuracy comparable to its native selector.

2 Related Work

KV cache compression and training-free sparse attention. KV cache compression discards cached entries to reduce the cache footprint [38, 18, 29, 28], which can cause accuracy loss when discarded context is later needed. QUEST and SparQ select tokens or blocks per query while retaining the full KV cache [25, 23]; RocketKV combines permanent eviction with query-aware sparse attention [4]. MInference [15], FlexPrefill [16], and XAttention [31] exploit dynamic sparse patterns to accelerate prefill. Methods retaining the full cache still face growing capacity demands.

Trainable sparse attention. Trainable sparse attention learns sparse behavior during pretraining or adaptation. InfLLM-V2 [39] and MoBA [21] use parameter-free block selection over mean-pooled keys; SeerAttention [11] learns a gate over pooled queries and keys through self-distillation. Native Sparse Attention (NSA) [34] fuses compressed, sparse, and sliding-window branches via learnable gating. DeepSeek Sparse Attention (DSA) [7] moves to token-level sparsity. DeepSeek-V4 [8] further interleaves Compressed Sparse Attention (CSA) with Heavily Compressed Attention (HCA), cutting per-token KV cache $\sim 10\times$ over DeepSeek-V3.2. However, its $8\times$ longer context still leaves absolute capacity a bottleneck. For reducing sparse selection overhead, IndexCache [3] reuses top-k indices across adjacent layers, and HISA [32] replaces the flat scan with hierarchical filtering. For block-sparse GQA, SparDA adapts DSA's decoupled-indexer design to one-layer-ahead selection with one Forecast head per GQA group.

Sparse attention with KV cache offloading. ArkVale [5] recalls evicted KV entries from CPU, while ShadowKV [24] combines low-rank key reconstruction with offloaded values. MagicPIG [6] uses locality-sensitive hashing to sample entries and computes their attention on CPU. NOSA [14] pairs a query-agnostic eviction head with a query-aware selector and issues UVA-based CPU-to-GPU transfers synchronously per layer, reducing volume rather than hiding latency. SparseServe [41] and HiSparse [30] offload KV caches to host memory. SparseServe uses working-set-aware batch sizing and fragmentation-aware transfers, while HiSparse combines LRU eviction with a fused fetch kernel. For models that share selections across layers, HiSparse also overlaps future-layer KV prefetch with earlier-layer computation. InfiniGen [17] is closest to SparDA. It uses the raw hidden state as a cross-layer proxy to prefetch a subset of CPU-resident KV to GPU before each layer. These DMA-based memory copies overlap with the preceding layer's execution. SparDA instead learns a cross-layer Forecast projection without requiring shared selections, bridging DSA's trained decoupled indexer with InfiniGen's lookahead. We co-design this projection with a persistent UVA prefetch kernel for communication-computation overlap.

3 Preliminary

3.1 InfLLM-V2

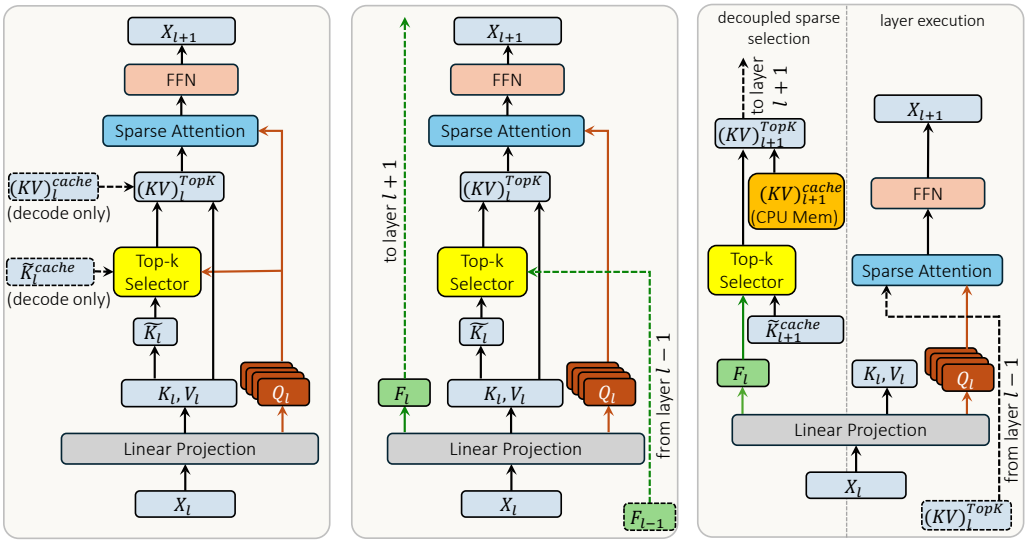
Our block-sparse implementation builds SparDA on top of InfLLM-V2 [39]. InfLLM-V2 decomposes attention into initial tokens, a local sliding window, and top-k selected blocks. SparDA modifies the top-k selection path while preserving the initial and local components. This section recaps the block-sparse components that SparDA directly extends; the native token-level DSA adaptation is described in Section 5.2.

Unified sparse attention. InfLLM-V2 merges selected attention and sliding-window attention into a single sparse attention module with shared KV projections $\mathbf{W}_K$ and $\mathbf{W}_V$ inherited from the pre-trained dense model. At layer l, the attended block set for a query token at position i is:

$$\mathcal{B}_l(i) = \mathcal{B}_{\text{init}} \cup \mathcal{B}_{\text{local}}(i) \cup \mathcal{B}_{\text{topk}}(i), \tag{1}$$

where $\mathcal{B}_{\text{init}}$ covers a fixed set of initial blocks, $\mathcal{B}_{\text{local}}(i)$ covers the local blocks, and $\mathcal{B}_{\text{topk}}(i)$ contains the top-k blocks selected by compression scores. Full attention is computed only over the KV entries in $\mathcal{B}_l(i)$, reducing the per-query attention cost from $O(T)$ to $O(|\mathcal{B}_l(i)| \cdot B)$, where T is the sequence length and B is the block size.

Three-stage block representation. InfLLM-V2 uses a parameter-free, coarse-to-fine compression pipeline to compute block-level relevance scores for $\mathcal{B}_{\text{topk}}$. In the first stage, keys $\mathbf{K}_l$ are mean-pooled



(a) Baseline (InfLLM-V2) (b) SparDA (Prefill) (c) SparDA (Decode)

Figure 2: SparDA architecture. **(a)** In the baseline (InfLLM-V2), $\mathbf{Q}_l$ drives both top-k selection and sparse attention within the same layer, so selection sits on the attention critical path. **(b, c)** SparDA adds a Forecast $\mathbf{F}_l$ to the linear projection and uses $\mathbf{F}_{l-1}$ from the previous layer to select blocks for the current layer. This decoupling pays off in two regimes: during prefill **(b)** the Forecast replaces the expensive multi-head selector with a Forecast indexer that uses one Forecast head per GQA group; during decode **(c)** $\mathbf{F}_l$ predicts $\mathcal{B}_{l+1}$ early enough to prefetch the selected KV blocks from CPU asynchronously, hiding the PCIe transfer behind the current layer's compute.

into overlapping compressed representations $\widetilde{\mathbf{K}}_l$ with kernel size l_{C_1} and stride s_{C_1}:

$$\widetilde{\mathbf{K}}_{l,j} = \text{Mean}(\mathbf{K}_{l,\, j\cdot s_{C_1}:j\cdot s_{C_1}+l_{C_1}}). \tag{2}$$

In grouped-query attention (GQA) [1], each group m contains G query heads that share one KV head. For head h in group m, scoring against the corresponding compressed keys yields per-compressed-key relevance $\mathbf{S}_{l,m}^h$. Scores are then summed within each group to produce a shared importance score $\mathbf{S}_{l,m}^{\text{shared}}$. In the final stage, a max-pooling operation over each $\mathbf{S}_{l,m}^{\text{shared}}$ produces the block-level score used for top-k selection. This three-stage process computes block scores from several overlapping sub-blocks rather than a single coarse pooling, preserving finer-grained information.

4 Method

4.1 SparDA Architecture

As discussed in Section 1, the primary goals of SparDA are twofold: (1) to enable KV cache prefetching that overlaps with layer execution for latency hiding; and (2) to reduce the overhead of sparse selection. SparDA achieves both through a single architectural change: a fourth per-layer projection, the *Forecast* $\mathbf{F}_l$, produced alongside the standard $\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l$, that drives a compact sparse-selection indexer. $\mathbf{F}_l$ decouples sparse selection from the attention query so it can be computed one layer ahead. This decoupling also means the indexer no longer needs the full query-head structure used for attention. In our GQA implementation, $\mathbf{F}_l$ replaces the original per-query-head scoring loop with one Forecast head per GQA group and skips the softmax normalization. Figure 2 illustrates the resulting layer structure compared with the InfLLM-V2 baseline. The formulation below describes this block-sparse GQA implementation; Section 5.2 and Appendix E describe the adaptation to native token-level DSA.

Decoupled sparse selection. In the original InfLLM-V2 sparse attention (Figure 2a), the query $\mathbf{Q}_l$ is used both for top-k sparse selection and for the sparse attention computation, and both steps

happen within the same layer. SparDA decouples these two roles. The linear projection ϕ_l in each layer produces an additional Forecast $\mathbf{F}_l$ besides the standard $\mathbf{Q}_l$, $\mathbf{K}_l$, $\mathbf{V}_l$:

$$(\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l, \mathbf{F}_l) = \phi_l(\mathbf{X}_l). \tag{3}$$

For brevity, the equations in this subsection drop the per-query-token index used in Section 3. The Forecast predicts the next layer's selection for the same token, so lookahead is across layers and leaves causal masking unchanged. $\mathbf{F}_l$ is used for top-k sparse selection while $\mathbf{Q}_l$ is used only for the sparse attention computation. Let $f_{\text{top}}(\cdot, k)$ denote the InfLLM-V2 block-selection operator, which max-pools compressed-key-grid scores to block-level scores and returns the top-k blocks. Specifically, $\mathbf{F}_l$ scores against the compressed keys $\widetilde{\mathbf{K}}_{l+1}$ of layer $l{+}1$, and the resulting top-k blocks are merged with the initial and local blocks to form the attended set for layer $l{+}1$:

$$\mathcal{B}_{l+1} = \mathcal{B}_{\text{init}} \cup \mathcal{B}_{\text{local}} \cup f_{\text{top}}\left(\mathbf{F}_l \widetilde{\mathbf{K}}_{l+1}^{\top},\ k\right). \tag{4}$$

For NOSA, the attended set additionally includes blocks chosen by its unchanged query-agnostic selector (Appendix C). The sparse attention at layer $l{+}1$ still uses the original query $\mathbf{Q}_{l+1}$:

$$\mathbf{O}_{l+1} = \text{Attn}(\mathbf{Q}_{l+1},\ \mathbf{K}_{l+1}[\mathcal{B}_{l+1}],\ \mathbf{V}_{l+1}[\mathcal{B}_{l+1}]). \tag{5}$$

For the first layer ($l{=}0$), where no previous Forecast exists, SparDA produces a current-layer Forecast $\mathbf{F}_0^{\text{cur}}$ from $\mathbf{X}_0$ via a separate projection and uses it for same-layer selection. $\mathbf{F}_0^{\text{cur}}$ still uses one Forecast head per GQA group rather than the original attention query, so the selection cost remains lower than the baseline's even without the one-layer-ahead benefit. For the final layer, $\mathbf{F}_{L-1}$ is unused since there is no subsequent layer to predict for.

The benefit of decoupled selection differs across inference phases. During prefill (Figure 2b), all keys are already on GPU, so the only saving is the reduced sparse selection cost. During decode (Figure 2c), the KV cache is offloaded to CPU memory. $\mathbf{F}_l$ produces $\mathcal{B}_{l+1}$ while layer l is still executing. This lets the runtime prefetch the selected KV entries from CPU and overlap the PCIe transfer with layer execution. The compressed keys $\widetilde{\mathbf{K}}_{l+1}^{\text{cache}}$ are kept on GPU and updated incrementally. Initial and local blocks are always attended and GPU-resident for both Sparse and SparDA, and are excluded from the predicted top-k budget. Newly appended keys lie in the local window, so nonlocal block selection does not require the next layer's current-token key. Algorithms 1 and 2 in Appendix A summarize the per-layer steps.

Forecast indexer. The key observation is that once Forecast is decoupled from the attention query, sparse selection no longer has to use the same head layout as attention. Block-sparse selectors such as InfLLM-V2 currently score each GQA group with all G query heads, because the attention query itself drives selection. DSA's lightning indexer [7] already exploits this decoupling at the token level with fewer heads than the attention query. SparDA brings the same design to block-sparse attention. $\mathbf{F}_l$ has one Forecast head per GQA group (one per KV head) rather than one per query head. This eliminates the per-query-head scoring loop and significantly reduces top-k selection overhead. It also naturally skips the softmax operation that standard block-sparse selectors apply before top-k ranking, since there is no need for score summation across multiple query heads within a GQA group.

4.2 Indexer Training

SparDA can be added to existing sparse-pretrained models by training only the Forecast projections, without retraining the base model. On both MiniCPM4.1-8B and NOSA-8B, these projections add only 33.5M parameters (0.41% of the 8B total), making SparDA a lightweight add-on with negligible model-size overhead.

Learning next-layer importance. The Forecast learns to estimate which KV entries will matter to the next layer. During training, the frozen model provides that layer's actual attention queries and keys, from which we construct a relevance target. The Forecast from the preceding layer learns to predict this target using the coarse compressed keys available at inference. For the two block-sparse models, we construct the target with finer key compression, so that small groups of relevant tokens are less diluted when forming the supervision signal. The target therefore differs from the Sparse baseline's own coarse selection scores. Only the Forecast projections are optimized.

Aligning the target and prediction. The finer target initially has more score positions than the Forecast prediction. We max-pool the target scores onto the prediction's compressed-key grid and normalize both to form comparable distributions. A separate pooling step converts the aligned target scores into block scores for top-k selection. Thus, target blocks are selected *after* pooling to the inference block size, while the loss retains the individual compressed-key scores within those blocks. This separates the choice of important blocks from the finer relevance information used to train the Forecast.

Training objective. Inspired by DSA [7], we minimize KL divergence from the target distribution to the Forecast prediction. For both distributions, we keep the compressed-key positions within the target-selected blocks individually and combine all remaining probability mass into one "rest" entry. Let $\bar{p}^{\text{tgt}}$ and $\bar{p}^{\text{pred}}$ denote these normalized distributions. The objective is

$$\mathcal{L}_{\text{KL}} = \mathbb{E}_{l,i,m}\left[\text{KL}\left(\bar{p}^{\text{tgt}}_{l,i,m} \,\|\, \bar{p}^{\text{pred}}_{l,i,m}\right)\right], \tag{6}$$

where the average is over layers l, query tokens i, and GQA groups m. The loss teaches the Forecast both the relative importance of selected regions and how much probability belongs elsewhere. Softmax is used to form distributions during training; inference only needs score rankings. Appendix B gives the pooling settings and training details, and Table 8 compares the target resolutions. The GLM adaptation uses its native DSA indexer as the teacher, as described in Section 5.2.

4.3 Efficient Implementation

At inference, SparDA's one-layer-ahead Forecast produces $\mathcal{B}_{l+1}$ while layer l is still executing. SparDA exploits this early availability through an asynchronous prefetch pipeline that transfers only the selected KV blocks from CPU to GPU before they are needed.

Persistent UVA kernel for asynchronous prefetch. Once the sparse pattern for the next layer is predicted, the runtime reuses GPU-resident blocks in $\mathcal{B}_{l+1}$ and fetches only the missing blocks from pinned CPU memory on a dedicated CUDA stream so that the transfer overlaps with layer execution. Instead of relying on many small, irregular memory copies, SparDA uses a persistent Triton [26] kernel based on Unified Virtual Addressing (UVA). The kernel keeps a small fixed set of GPU thread blocks, or Cooperative Thread Arrays (CTAs), active and lets them continuously process block-transfer tasks within a single launch, reducing launch overhead, avoiding frequent synchronization, and limiting interference with the main compute stream. As a result, CPU-to-GPU KV cache transfers can be largely overlapped with layer execution, making prefetching practical even with large-batch, high-throughput decoding.

Batch-adaptive CTA allocation. The number of CTAs in the persistent kernel controls the trade-off between prefetch throughput and layer execution speed. More CTAs accelerate transfers toward the PCIe bandwidth ceiling but consume Streaming Multiprocessors (SMs) that would otherwise run the attention and FFN kernels. At small batch sizes the GPU is underutilized and layer execution dominates the critical path; a modest CTA budget suffices because the total transfer volume is small. As batch size grows, layer execution time scales sub-linearly (better SM occupancy) while prefetch volume grows linearly. Beyond a crossover point the prefetch becomes the new bottleneck, and allocating additional CTAs is worthwhile despite the marginal slowdown in layer execution. SparDA therefore adopts a simple heuristic to choose the CTA count that maximizes overall throughput for the underlying hardware. The sweep and chosen configurations are reported in Table 9 (Appendix D).

5 Experiments

5.1 Experimental Setup

Our main accuracy and efficiency evaluation uses two 8B models: MiniCPM4.1-8B [22], whose sparse backbone is InfLLM-V2 [39], and NOSA-8B [14], which adds a query-agnostic eviction head on top of InfLLM-V2. Accuracy is measured on HELMET [33], LongBench [2], RULER [13], and a long-reasoning suite (MATH-500 [19], AIME 2024 [36], AIME 2025 [37]). We compare four configurations on these models: dense, sparse, InfiniGen [17], and SparDA. We additionally evaluate architectural transfer on GLM-5.1 [35], comparing accuracy against its native token-level DSA

selector. We do not directly compare against DSA-specific accelerators such as IndexCache [3] or HISA [32]. Efficiency is measured for the 8B models on NVIDIA H100 and A100 GPUs using the runtime described in Section 4.3. Dense† and Sparse† denote the no-offload configurations. Sparse, InfiniGen, and SparDA use CPU offloading. The main text reports H100 results; Appendix D includes A100 results. Evaluation details for the 8B models and GLM-5.1 are in Appendices C and E, respectively.

5.2 Accuracy Results

Aggregate accuracy. Across the four benchmark families, the results support accuracy comparable to the full-query Sparse baseline (Table 1). On MiniCPM4.1-8B, SparDA reaches an average of 61.9 versus 61.7 for Sparse, with reasoning scores of 85.7 versus 84.5, a higher RULER score (+0.5), LongBench essentially flat, and HELMET slightly lower (-0.6). On NOSA-8B, the averages are 51.1 versus 49.3, with reasoning scores of 54.8 versus 50.5, higher RULER (+1.7) and HELMET (+1.2) scores, and slightly lower LongBench (-0.1). All four configurations use the eight-seed AIME means reported in Appendix C (Table 7) for their reasoning averages. SparDA's positive AIME mean differences over Sparse are within the reported seed-to-seed variation, and we do not claim statistically significant gains from these runs. Across benchmark families and individual tasks, the observed differences can be positive or negative. Dense scores remain higher than both Sparse and SparDA on HELMET and RULER at the models' native maximum sequence lengths (64K for MiniCPM4.1-8B, 32K for NOSA-8B). InfiniGen also has lower HELMET and RULER scores than Sparse on both models. LongBench differences are mixed. Its reasoning average is tied on MiniCPM at the displayed precision and higher on NOSA. Per-task breakdowns are in Appendix D.

Table 1: Aggregated benchmark averages. Reasoning averages the single-run MATH-500 score and eight-seed AIME 2024/2025 means for every method (Table 7). *Avg* is the arithmetic mean of the four benchmark families, computed before rounding. Best values among sparse methods (excluding Dense) are **bolded**.

	MiniCPM4.1-8B					NOSA-8B				
Method	HELMET	LongBench	RULER	Reasoning	Avg	HELMET	LongBench	RULER	Reasoning	Avg
Dense	41.7	44.8	85.3	85.9	64.4	39.3	42.5	86.2	52.0	55.0
Sparse	**38.9**	45.0	78.2	84.5	61.7	32.2	**42.4**	72.2	50.5	49.3
InfiniGen	33.5	**45.1**	68.4	84.5	57.9	28.1	41.6	65.2	54.4	47.3
SparDA	38.3	**45.1**	**78.7**	**85.7**	**61.9**	**33.4**	42.3	**73.9**	**54.8**	**51.1**

Length generalization. SparDA has higher measured RULER scores than Sparse at each evaluated sequence length on both models (Table 2). On NOSA-8B, the observed gap widens steadily from +1.7 at 32K to +4.3 at 128K; MiniCPM4.1-8B shows a more mixed pattern (+1.5 at 32K, +0.5 at 64K, +2.1 at 96K, +1.1 at 128K). These measurements support using the Forecast at the evaluated extended contexts; they do not establish a general accuracy advantage across tasks.

Table 2: RULER average accuracy at extended sequence lengths. SparDA has higher measured scores at all evaluated lengths on both models; the observed gap widens steadily with sequence length on NOSA-8B.

	MiniCPM4.1-8B				NOSA-8B			
Method	32K	64K	96K	128K	32K	64K	96K	128K
Sparse	86.1	78.2	68.7	67.7	72.2	56.6	48.8	40.7
SparDA	**87.6**	**78.7**	**70.8**	**68.8**	**73.9**	**60.5**	**52.9**	**45.0**
Δ	+1.5	+0.5	+2.1	+1.1	+1.7	+3.9	+4.1	+4.3

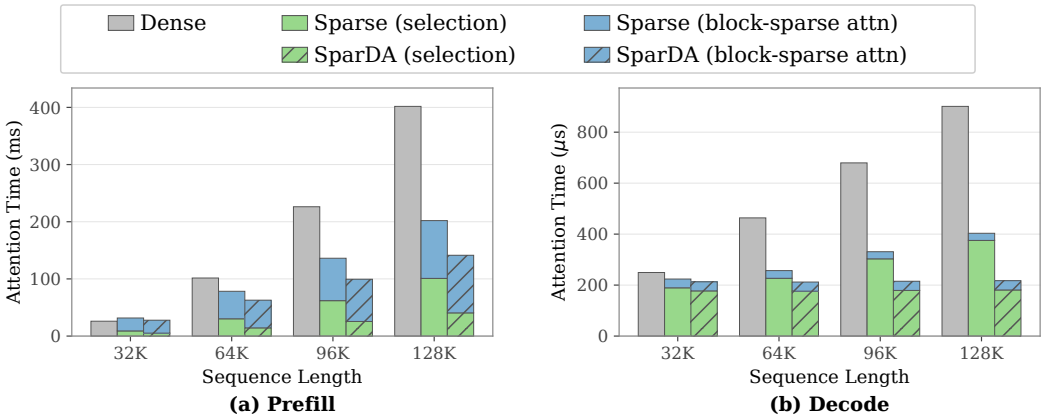


Figure 3: Per-layer attention wall time on MiniCPM4.1-8B at batch size 4. **(a)** Prefill (ms). **(b)** Decode (μs). Each group contains a Dense bar (gray) and Sparse/SparDA bars (solid/hatched) stacked by block selection (green) and block-sparse attention (blue). SparDA significantly reduces block-selection cost for both prefill and decode.

Transfer to token-level DSA. To test transfer beyond 8B block-sparse models, we adapt SparDA to GLM-5.1, a 754B mixture-of-experts model with native token-level DSA [35, 12]. We freeze the backbone and train the Forecast at 64K context against the native DSA indexer's selection distribution, rather than the full attention query. The Forecast retains the native indexer's 32 heads, each with dimension 128. This indexer is already compact relative to the model's 64 attention heads. Reducing the Forecast head count further degraded accuracy. This adaptation therefore does not provide the prefill selection-compute savings of our GQA implementation. For GLM-5.1, we use the Forecast during decoding and native DSA during prefill. Across RULER, HELMET, LongBench, and reasoning, SparDA obtains an average score of 80.20 versus 80.10 for native DSA (Table 3), supporting comparable accuracy at this larger scale and with token-level selection. Appendix E gives the evaluation settings and detailed results. These results support architectural transfer with comparable accuracy.

Table 3: GLM-5.1 aggregate accuracy. HELMET, LongBench, and RULER use the 64K setting; reasoning uses a 32K generation budget. Reasoning averages MATH-500 and the eight-seed AIME means. *Avg* averages the four benchmark families. Best values are **bolded**.

	GLM-5.1 (754B MoE)				
Method	HELMET	LongBench	RULER	Reasoning	Avg
Native DSA	**71.07**	**55.52**	**96.46**	97.37	80.10
SparDA	70.58	55.40	96.42	**98.41**	**80.20**

5.3 Efficiency Results

Attention breakdown. To understand where the speedup comes from, we break down per-layer attention time on MiniCPM4.1-8B at batch size 4 into block selection (green) and block-sparse attention (blue) for Sparse and SparDA (Figure 3). We focus on MiniCPM4.1-8B because NOSA-8B includes additional query-agnostic components that obscure the comparison of query-aware sparse attention, the core component shared by most sparse attention methods. During prefill (Figure 3a), both block-sparse attention and block selection grow with sequence length, with selection becoming comparable to attention at 128K. SparDA reduces block-selection cost by approximately 2.5$\times$ at 128K while keeping block-sparse attention time comparable to Sparse. During decode (Figure 3b), block-sparse attention is cheap because each step has only one query token. Block selection becomes the bottleneck for Sparse, and its cost grows with sequence length. SparDA's Forecast indexer keeps decode-time selection nearly flat, cutting the overhead by more than 2$\times$ at 128K. The near-flat decode profile reflects that the indexer's compute is small enough to leave the GPU under-utilized, so longer contexts barely affect decode latency.

Prefill throughput. The selection speedup translates directly into prefill throughput gains (Table 4). We report a single batch size of 4 because prefill throughput is insensitive to batch size, as GPUs are already fully utilized during the compute-heavy prefill phase. Dense attention leads at short sequences but its quadratic scaling causes it to fall behind as sequence length grows. SparDA consistently outperforms all other methods from 64K onward on MiniCPM4.1-8B, achieving up to 1.25× speedup over Sparse and 2.11× over Dense at 128K. On NOSA-8B, SparDA leads from 96K onward with up to 1.16× over Sparse and 1.40× over Dense at 128K. All sparse methods are slower on NOSA-8B because its query-agnostic eviction head adds extra work. The main source of SparDA's speedup is the reduced block-selection cost of its Forecast indexer (Figure 3a), which scales more favorably than the baseline's multi-head selector as sequence length increases. KV cache offloading itself has negligible impact on prefill throughput. Sparse† (no offload) and Sparse (with offload) achieve nearly identical throughput. The only offload-time transfer is the asynchronous CPU writeback of newly created KV cache entries after they are used for attention.

Table 4: Prefill throughput (tok/s) on NVIDIA H100 GPU at batch size 4. Sequence length swept from 32K to 128K. †: no offload. Best value per column is **bolded**.

	MiniCPM4.1-8B				NOSA-8B			
Method	32K	64K	96K	128K	32K	64K	96K	128K
Dense†	**20388.3**	13673.7	10228.3	8085.8	**20438.9**	**13701.0**	10244.1	8118.0
Sparse†	18706.6	16377.3	14774.1	13676.1	12896.5	11418.0	10514.7	9832.5
Sparse	18548.4	16254.4	14707.7	13661.8	12778.3	11359.2	10448.5	9805.2
InfiniGen	18445.3	16249.9	14749.2	13643.9	12838.5	11372.4	10501.6	9749.3
SparDA	19845.6	**18379.5**	**17715.2**	**17087.6**	13456.0	12386.1	**11807.3**	**11332.7**

Decode throughput. Decode is where the lookahead design pays off most, as it lets CPU-to-GPU KV prefetch overlap with the current layer's compute. Within the offload regime at 128K, SparDA achieves up to 1.69× speedup over Sparse on MiniCPM4.1-8B and 1.40× on NOSA-8B (Table 5), driven by reduced block-selection overhead (Figure 3b) and overlapped prefetch that hides the otherwise synchronous CPU-to-GPU transfer. The largest speedups appear at middle batch sizes (B8–B16), where prefetch and layer execution are roughly balanced. Against the non-offload baselines (†), which OOM past B16 at long contexts, offloading lets SparDA run much larger feasible batches and obtain correspondingly larger throughput gains, up to $5.28\times$ over Sparse† and $9.21\times$ over Dense† on MiniCPM4.1-8B. Against the offload baseline, NOSA-8B shows lower speedups because its query-agnostic eviction head already reduces KV fetch traffic, leaving less room for overlap. InfiniGen is consistently slower than Sparse primarily because it gathers top-k KV blocks on CPU before transferring them to GPU, making CPU-side gather the bottleneck. Because SparDA retains the layer-0 KV cache on GPU (Appendix C), it OOMs slightly earlier than Sparse at long contexts. Nevertheless, SparDA's peak throughput exceeds Sparse's at every context length, even when Sparse runs at a larger batch.

Appendix D provides a decode speedup breakdown that separates block-selection savings from prefetch overlap, ablations on the compression window and prefetch CTA allocation, and throughput results on NVIDIA A100 GPU.

6 Limitations

SparDA builds on an existing sparse attention backbone to improve inference efficiency. The decoupled indexer retains the base model and sparse attention computation, but it can change which KV entries are selected. Its accuracy therefore depends on both the base model and the Forecast's selection decisions. Our GLM-5.1 results extend the accuracy evaluation to token-level DSA at 754B scale; efficiency measurements remain limited to the two 8B models. We have not measured SparDA's throughput benefit on GLM-5.1 because integrating KV offloading and lookahead prefetch into its multi-node serving stack requires substantial additional engineering. GLM-5.1 is evaluated in SGLang across four nodes of eight H100 GPUs, with tensor, pipeline, and expert parallelism, whereas our prefetch runtime currently targets one GPU per replica. Coordinating per-rank KV partitions, pipeline-stage boundaries, and expert all-to-all communication makes this integration complex. Measuring the resulting throughput and extending the method to other sparse architectures, including the Compressed Sparse Attention (CSA) path of DeepSeek-V4 [8], remain future work.

Table 5: Decode throughput (tok/s) on NVIDIA H100 GPU. Batch size (B) swept from 4 to 128. †: no offload. "–": OOM. Best value per column is **bolded**.

		MiniCPM4.1-8B						NOSA-8B					
Context	Method	B4	B8	B16	B32	B64	B128	B4	B8	B16	B32	B64	B128
32K	Dense†	248.3	352.2	440.2	–	–	–	**252.0**	350.7	439.7	–	–	–
	Sparse†	**277.2**	**550.9**	**1073.0**	–	–	–	241.3	**488.1**	**947.1**	–	–	–
	Sparse	224.4	420.9	661.5	861.8	1088.4	1225.1	244.0	478.0	795.8	1244.9	1861.6	2145.5
	InfiniGen	56.3	75.1	98.8	140.4	184.8	222.4	82.0	120.3	166.0	205.9	269.9	356.5
	SparDA	235.7	471.9	918.5	**1205.2**	**1328.0**	**1358.2**	214.7	433.2	873.2	**1579.9**	**2106.6**	**2312.1**
64K	Dense†	174.1	218.3	–	–	–	–	174.3	217.6	–	–	–	–
	Sparse†	**246.5**	466.2	–	–	–	–	**233.2**	**439.7**	–	–	–	–
	Sparse	213.1	350.8	549.4	756.1	937.2	1041.9	221.0	397.9	704.6	1071.1	1553.1	1793.1
	InfiniGen	55.2	63.3	94.0	129.1	166.5	–	78.1	106.2	139.1	181.0	220.4	–
	SparDA	234.9	**470.3**	**811.8**	**1026.5**	**1049.9**	**1159.1**	218.8	431.2	**870.9**	**1392.3**	**1905.2**	**1996.3**
96K	Dense†	133.8	159.0	–	–	–	–	130.0	158.3	–	–	–	–
	Sparse†	215.5	401.4	–	–	–	–	204.1	380.1	–	–	–	–
	Sparse	185.7	306.0	489.0	679.4	839.5	**934.3**	191.5	341.3	605.5	962.4	1354.3	**1580.5**
	InfiniGen	49.4	64.9	86.0	117.9	–	–	75.1	104.0	129.1	165.2	–	–
	SparDA	**238.6**	**475.6**	**752.9**	**939.2**	**1076.3**	–	**215.5**	**437.6**	**852.9**	**1263.7**	**1694.1**	–
128K	Dense†	108.6	–	–	–	–	–	108.4	–	–	–	–	–
	Sparse†	189.5	–	–	–	–	–	179.3	–	–	–	–	–
	Sparse	167.8	279.5	447.9	618.6	788.9	–	173.2	285.4	529.2	898.7	1298.3	–
	InfiniGen	51.8	66.5	85.6	117.5	–	–	77.5	105.2	131.7	166.9	–	–
	SparDA	**240.2**	**471.2**	**705.3**	**899.2**	**1000.1**	–	**219.0**	**399.3**	**735.0**	**1127.0**	**1463.3**	–

7 Conclusion

SparDA shows that sparse selection can be treated as a trainable, schedulable signal rather than an operation tied to the current attention query. By decoupling selection from attention, the Forecast projection predicts block choices one layer ahead, letting the runtime overlap CPU-to-GPU KV transfer with current-layer execution while reducing selector cost through a compact GQA-level indexer. More broadly, the results suggest that sparse attention mechanisms should expose future memory-access patterns early enough for the serving system to act on them, turning sparsity from a compute-saving device into an offloading-friendly schedule. On the two evaluated 8B models, SparDA maintains accuracy comparable to the full-query sparse-attention baseline while improving long-context inference efficiency without retraining the base model. Our adaptation to GLM-5.1 (754B MoE) also achieves accuracy comparable to its native token-level selector. This provides evidence that decoupled selection transfers beyond block-sparse 8B models.

Acknowledgments

This work was funded by NVIDIA. Yaosheng Fu, Song Han, and Oreste Villa are employed by NVIDIA, and Song Han is also affiliated with MIT. Guangxuan Xiao and Xin Dong contributed to this work while employed at NVIDIA and are now with Thinking Machines Lab and ByteDance Seed, respectively. The authors declare no other competing interests.

References

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[2] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL) (Volume 1: Long Papers)*, 2024.

[3] 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.

[4] Payman Behnam, Yaosheng Fu, Ritchie Zhao, Po-An Tsai, Zhiding Yu, and Alexey Tumanov. RocketKV: Accelerating long-context LLM inference via two-stage KV cache compression. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[5] Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. ArkVale: Efficient generative LLM inference with recallable key-value eviction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[6] Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. MagicPIG: LSH sampling for efficient LLM generation. In *International Conference on Learning Representations (ICLR)*, 2025.

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

[8] DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

[9] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. LongRoPE: Extending LLM context window beyond 2 million tokens. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[10] Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL) (Volume 1: Long Papers)*, 2025.

[11] Yizhao Gao, Zhichen Zeng, DaYou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden So, Ting Cao, Fan Yang, and Mao Yang. SeerAttention: Self-distilled attention gating for efficient long-context prefilling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

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

[13] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

[14] Yuxiang Huang, Pengjie Wang, Jicheng Han, Weilin Zhao, Zhou su, Ao Sun, Hongya Lyu, Hengyu Zhao, Yudong Wang, Chaojun Xiao, Xu Han, and Zhiyuan Liu. NOSA: Native and offloadable sparse attention. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2026.

[15] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[16] Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. FlexPrefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *International Conference on Learning Representations (ICLR)*, 2025.

[17] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024.

[18] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[19] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024.

[20] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[21] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Yutao Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. MoBA: Mixture of block attention for long-context LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[22] MiniCPM Team. MiniCPM4: Ultra-efficient LLMs on end devices. *arXiv preprint arXiv:2506.07900*, 2025.

[23] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[24] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[25] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[26] Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, 2019.

[27] Siyuan Wang, Gaokai Zhang, Li Lyna Zhang, Ning Shang, Fan Yang, Dongyao Chen, and Mao Yang. LoongRL: Reinforcement learning for advanced reasoning over long contexts. In *International Conference on Learning Representations (ICLR)*, 2026.

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

[29] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations (ICLR)*, 2025.

[30] Zhiqiang Xie, Zhangheng Huang, Tingwei Huang, Ziyi Xu, Ruiyang Ma, and Christos Kozyrakis. HiSparse: Scaling sparse-attention decoding with hierarchical KV cache management. *arXiv preprint arXiv:2608.07009*, 2026.

[31] Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. XAttention: Block sparse attention with antidiagonal scoring. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

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

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

[34] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL) (Volume 1: Long Papers)*, 2025.

[35] Z.ai. GLM-5.1 model card. `https://huggingface.co/zai-org/GLM-5.1`, 2026.

[36] Yifan Zhang and Math-AI Team. American invitational mathematics examination (AIME) 2024. `https://huggingface.co/datasets/math-ai/aime24`, 2024. Dataset release; original competition problems copyright Mathematical Association of America.

[37] Yifan Zhang and Math-AI Team. American invitational mathematics examination (AIME) 2025. `https://huggingface.co/datasets/math-ai/aime25`, 2025. Dataset release; original competition problems copyright Mathematical Association of America.

[38] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[39] Weilin Zhao, Zihan Zhou, Zhou su, Chaojun Xiao, Yuxuan Li, Yanghao Li, Yudi Zhang, Weilun Zhao, Zhen Li, Yuxiang Huang, Ao Sun, Xu Han, and Zhiyuan Liu. InfLLM-V2: Dense-sparse switchable attention for seamless short-to-long adaptation. In *International Conference on Learning Representations (ICLR)*, 2026.

[40] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[41] Qihui Zhou, Peiqi Yin, Pengfei Zuo, Chao Wang, and James Cheng. SparseServe: Unlocking parallelism for dynamic sparse attention in long-context LLM serving. In *Proceedings of the 40th ACM International Conference on Supercomputing (ICS)*, 2026.

[42] Yufan Zhuang, Xiaodong Yu, Jialian Wu, Ximeng Sun, Ze Wang, Jiang Liu, Yusheng Su, Jingbo Shang, Zicheng Liu, and Emad Barsoum. Self-taught agentic long context understanding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL) (Volume 1: Long Papers)*, 2025.

A Algorithm Pseudocode

Algorithm 1: SparDA prefill step (all tokens, one layer)

Input: Hidden states $\mathbf{X}_l$; Forecast $\mathbf{F}_{l-1}$ from prior layer
Output: Hidden states $\mathbf{X}_{l+1}$; Forecast $\mathbf{F}_l$ for next layer

$(\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l, \mathbf{F}_l) \leftarrow \phi_l(\mathbf{X}_l)$ `// F_l is the Forecast`
$\mathcal{B}_l \leftarrow \mathcal{B}_{\text{init}} \cup \mathcal{B}_{\text{local}} \cup f_{\text{top}}(\mathbf{F}_{l-1}\, \widetilde{\mathbf{K}}_l^{\top},\ k)$ `// pooled block selection`
$\mathbf{O}_l \leftarrow \text{Attn}(\mathbf{Q}_l,\ \mathbf{K}_l[\mathcal{B}_l],\ \mathbf{V}_l[\mathcal{B}_l])$
$\mathbf{X}_{l+1} \leftarrow \text{FFN}_l(\mathbf{O}_l)$
return $\mathbf{X}_{l+1},\ \mathbf{F}_l$

For the first layer ($l{=}0$), where no previous Forecast exists, SparDA uses the layer-0 Forecast $\mathbf{F}_0^{\text{cur}}$ described in Section 4 for same-layer selection. In decode, the layer-0 KV cache remains on GPU, so the prefetch path begins from subsequent layers. At the final layer ($l{=}L{-}1$), the next-layer selection and prefetch steps in Algorithm 2 are skipped.

Algorithm 2: SparDA decode step (one token, one layer)

Input: Hidden state $\mathbf{X}_l$; selected blocks $\mathcal{B}_l$ and prefetched KV entries from prior layer
Output: Hidden state $\mathbf{X}_{l+1}$; selected blocks $\mathcal{B}_{l+1}$ for the next layer

$(\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l, \mathbf{F}_l) \leftarrow \phi_l(\mathbf{X}_l)$ `// F_l is the Forecast`
Append $(\mathbf{K}_l, \mathbf{V}_l)$ to $(\mathbf{K}_l^{\text{cache}}, \mathbf{V}_l^{\text{cache}})$
Update $\widetilde{\mathbf{K}}_l^{\text{cache}}$ incrementally from $\mathbf{K}_l^{\text{cache}}$
$\mathcal{B}_{l+1} \leftarrow \mathcal{B}_{\text{init}} \cup \mathcal{B}_{\text{local}} \cup f_{\text{top}}(\mathbf{F}_l\, \widetilde{\mathbf{K}}_{l+1}^{\text{cache}\top},\ k)$ `// one-layer-ahead pooled block selection`
Launch prefetch of missing blocks in $\mathcal{B}_{l+1}$: CPU→GPU on prefetch stream
Wait for the KV entries of $\mathcal{B}_l$ (if not yet complete) `// prefetch launched during layer` $l{-}1$
$\mathbf{O}_l \leftarrow \text{Attn}(\mathbf{Q}_l,\ \mathbf{K}_l^{\text{cache}}[\mathcal{B}_l],\ \mathbf{V}_l^{\text{cache}}[\mathcal{B}_l])$
$\mathbf{X}_{l+1} \leftarrow \text{FFN}_l(\mathbf{O}_l)$
return $\mathbf{X}_{l+1},\ \mathcal{B}_{l+1}$

B Training Details

Target construction and alignment. For the block-sparse models, mean-pool the frozen backbone's keys with kernel size 2 and stride 1 to form the training target, and with kernel size 32 and stride 16 for the Forecast prediction. The target uses the actual queries of the layer being predicted: apply causal softmax to each query head's scores and sum within each GQA group. Max-pool these fine target scores onto the prediction's $(32, 16)$ compressed-key grid, then normalize. The Forecast produces a distribution on that same grid using its own scores and causal softmax. Key compression, alignment of target scores, and pooling scores into final KV blocks are distinct operations.

Block selection and loss. To choose the target top-k blocks, max-pool the aligned target scores to the 64-token block granularity. Initial and local blocks are always included in attention and excluded from this top-k choice. For the loss, return to the shared compressed-key grid and retain each score position within the selected blocks. Apply the same selection to target and prediction, combine the remaining mass into one rest entry, and normalize before taking $\text{KL}(\text{target} \,\|\, \text{prediction})$. The loss therefore has one entry per retained compressed-key position plus the rest entry, rather than one entry per selected block. The backbone and target remain fixed; gradients update only the Forecast projections. Layer 0 uses the same-layer Forecast defined in Section 4.

Table 6 lists the indexer training hyperparameters used for both the MiniCPM4.1-8B and NOSA-8B experiments. Only the Forecast projection weights are trained; all backbone parameters remain frozen. Training uses 32 NVIDIA H100 GPUs; MiniCPM4.1-8B at 64K completes within 48 hours, and NOSA-8B at 32K completes within 24 hours.

Table 6: Indexer training hyperparameters.

Hyperparameter	Value
Optimizer	AdamW [20] (β_1=0.9, β_2=0.95, ϵ=10^{-6})
Weight decay	0.01
Learning rate	5×10^{-4} (constant)
Training steps	2 000 optimizer steps
Effective batch size	32 (accumulated across GPUs)
Precision	BF16 mixed precision
Gradient clipping	Max norm 0.5
Training data	ProLong-64K [10]
Sequence length (MiniCPM4.1-8B)	65 536 tokens
Sequence length (NOSA-8B)	32 768 tokens
Compression window	Target (l_{C_1}=2, s_{C_1}=1); prediction (l_{C_1}=32, s_{C_1}=16)

C Evaluation Details

Model configuration. Both MiniCPM4.1-8B and NOSA-8B use a block size of 64 tokens, 1 initial block, and a compression kernel of size 32 with stride 16 for mean-pooled keys. Both models use a total attended-block budget of 96, including the forced initial and local blocks. The k in the method equations counts only query-aware predicted blocks, excluding these forced blocks. MiniCPM4.1-8B uses a local window of 2048 tokens (32 blocks) following its default sparse configuration, with the remaining budget after initial/local blocks assigned to query-aware selection. NOSA-8B uses a local window of 1024 tokens (16 blocks) and allocates 24 blocks (25% of 96) for query-aware selection. Its query-agnostic eviction head fills the remaining budget, following Huang et al. [14]. For MiniCPM4.1-8B, the thinking mode is disabled for all benchmarks except the reasoning suite. For the SparDA indexer, the trained Forecast projections (the main projection and the layer-0 same-layer projection) are loaded from the checkpoint. All other model parameters are identical to the sparse baseline.

Long-context extension. MiniCPM4.1-8B natively supports up to 64K tokens and NOSA-8B up to 32K. For benchmarks that exceed each model's native maximum, we apply positional-encoding extension at inference time. For MiniCPM4.1-8B we use the official 128K-validated LongRoPE [9] frequency factors provided with the model checkpoint, which replace both the long-factor and short-factor tables in the rotary embedding. For NOSA-8B we increase the RoPE base frequency (`rope_theta`) from the default 10,000 to 40,000 following Huang et al. [14], a simple scaling that extends the effective context window to 128K. The same positional extension is applied identically across all attention configurations (Dense, Sparse, InfiniGen, SparDA) so that any accuracy differences are attributable solely to the attention method, not the positional encoding.

InfiniGen adaptation. InfiniGen [17] is originally a token-level sparse attention method. To integrate it into the block-sparse framework of InfLLM-V2, we max-pool InfiniGen's approximate attention scores along the key dimension with a kernel size equal to the block size, converting them to block-level scores. The resulting scores use the same model-specific budget allocation within 96 attended blocks and the same block-sparse attention kernel as Sparse and SparDA. Since the original InfiniGen does not accelerate prefill, we use InfLLM-V2 sparse prefill for InfiniGen and additionally build its partial key cache during prefill by projecting full post-RoPE keys onto a subset of dimensions for decode-time scoring.

HELMET. We evaluate at each model's native maximum (64K for MiniCPM4.1-8B, 32K for NOSA-8B) across all seven HELMET categories (recall, RAG, ICL, rerank, citation, long QA, summarization). Each category uses 100 test samples. Decoding is greedy (`temperature`=0, `do_sample`=`False`). Generation length varies per category (50–200 tokens). For tasks originally designed to be judged by GPT-4, we use GPT-5.2 as the judge.

LongBench. We evaluate all 14 English tasks at the native maximum. Decoding is greedy with task-specific maximum generation lengths (32–512 tokens depending on the task).

RULER. We evaluate all 13 synthetic tasks with 50 examples per task. Default sequence length is the native maximum; extended evaluations use 32K, 64K, 96K, and 128K. Decoding is greedy with task-specific generation limits (30–128 tokens).

Reasoning (MATH-500, AIME 2024, AIME 2025). MATH-500 contains 500 problems; AIME 2024 and AIME 2025 each contain 30 problems. Unlike the other benchmarks, reasoning uses sampling (`temperature`$=0.9$, `top_p`$=0.95$ for MiniCPM4.1-8B; `temperature`$=1.0$, `top_p`$=1.0$ for NOSA-8B) with a long generation budget (up to 64K tokens for MiniCPM4.1-8B, 8K for NOSA-8B) to allow chain-of-thought reasoning. NOSA-8B does not support thinking mode, so we use a two-shot prompt that demonstrates step-by-step reasoning via in-context learning, following Huang et al. [14]. Final answers are evaluated by a GPT-5.2 judge.

Multi-seed AIME evaluation. Each AIME dataset contains only 30 problems, making single-run scores sensitive to sampling variation. We therefore evaluate Dense, Sparse, InfiniGen, and SparDA on AIME 2024 and AIME 2025 with eight sampling seeds (1–8) for each model, using the settings above. Sparse and SparDA use paired seeds; the Dense and InfiniGen runs also share their per-problem sampling seeds. Table 7 reports avg@8 $\pm$ standard deviation across per-seed accuracies, with one sampled answer per problem per run. Thus, avg@8 averages single-attempt accuracies rather than measuring whether any of eight answers is correct. Dense and InfiniGen standard deviations use the population convention over the eight runs.

We use these AIME means throughout for all four configurations, including the per-dataset scores, reasoning averages, and overall averages. Reasoning is the arithmetic mean of the MATH-500 score and the two AIME means, with rounding applied after aggregation. MATH-500 retains its single-run scores. The $(32, 16)$ target-window ablation also retains single-run AIME scores; the $(2, 1)$ variant is the main SparDA configuration and uses the eight-seed means.

Relative to Sparse, SparDA has a higher mean and lower seed-to-seed standard deviation in all four model–dataset comparisons. Dense has the highest AIME means on MiniCPM4.1-8B, while SparDA has the highest on NOSA-8B. Given the observed sampling variation, we interpret the Sparse–SparDA comparisons as supporting comparable accuracy and do not claim statistically significant gains from these eight runs.

Table 7: AIME accuracy (GPT-5.2 judge, %), reported as avg@8 $\pm$ standard deviation over eight sampling seeds for every method. Each run evaluates all 30 problems per dataset.

Model	Method	AIME 2024	AIME 2025
MiniCPM4.1-8B	Dense	84.6 ± 4.1	76.3 ± 4.2
	Sparse	83.8 ± 4.9	72.1 ± 7.9
	InfiniGen	82.9 ± 3.5	72.9 ± 3.9
	SparDA	84.2 ± 4.6	75.4 ± 5.6
NOSA-8B	Dense	55.8 ± 7.0	32.1 ± 5.5
	Sparse	49.2 ± 10.4	30.0 ± 9.4
	InfiniGen	54.6 ± 9.1	35.8 ± 7.2
	SparDA	56.7 ± 5.6	36.2 ± 8.4

Efficiency. Throughput is measured on two GPU configurations: (1) NVIDIA H100 GPU with 80 GB HBM3 memory and PCIe Gen5$\times$16 CPU-GPU connection, and (2) NVIDIA A100 GPU with 80 GB HBM2e memory and PCIe Gen4$\times$16 CPU-GPU connection. Both systems have 2 TB of CPU memory. For each throughput setting, we discard one warmup run and report the average of two measured runs. All methods in the 8B throughput evaluation are implemented on top of the high-throughput NOSI inference engine from Huang et al. [14]. NOSI caches the top-k KV blocks used by the previous decoding step for each layer by default, so each decode step only needs to fetch the newly selected KV blocks that were not in the previous step's selection. For offloading, pinned CPU memory stores the full KV history; the GPU retains compressed keys $\widetilde{\mathbf{K}}_l$ ($\approx 1/s_{C_1}$ the size of $\mathbf{K}_l$), initial/local blocks, and cached selected blocks. SparDA additionally keeps the full layer-0 KV on GPU because there is no prior Forecast to prefetch it. In the 32-layer GQA configuration, this costs about 1 KB per token in FP16/BF16, or about 8 GB at 128K context and batch 64 (1/32 of the

full KV cache), explaining the earlier OOM at large context–batch settings. Layer 0 could instead be offloaded at the cost of a transfer without one-layer-ahead overlap.

D Additional Results

Compression window ablation. The finer $(l_{C_1}{=}2, s_{C_1}{=}1)$ target compression window used during indexer training (Section 4.2) yields higher measured aggregate scores on both models (Table 8). On MiniCPM4.1-8B, its RULER score is higher by +3.0, while HELMET and LongBench are nearly unchanged. On NOSA-8B, its HELMET, LongBench, and RULER scores are higher, including +2.5 on RULER, while the reasoning score is 54.8 versus 55.4 for the coarser window. The fine-window reasoning scores use eight-seed AIME means, whereas the coarse-window scores use single runs, so the reasoning differences should be interpreted cautiously. These long-context benchmark results motivate our use of the finer target window.

Table 8: Ablation on the target compression window (l_{C_1}, s_{C_1}) for indexer training. The prediction window is fixed at $(32, 16)$. Reasoning uses eight-seed AIME means for the $(2, 1)$ variant and single-run scores for $(32, 16)$; MATH-500 is single-run for both. *Avg* is the arithmetic mean of the four benchmark families, computed before rounding. Best values per column are **bolded**.

	MiniCPM4.1-8B					NOSA-8B				
Target window	HELMET	LongBench	RULER	Reasoning	Avg	HELMET	LongBench	RULER	Reasoning	Avg
$(32, 16)$	38.2	**45.1**	75.7	82.5	60.4	31.6	42.0	71.4	**55.4**	50.1
$(2, 1)$	**38.3**	**45.1**	**78.7**	**85.7**	**61.9**	**33.4**	**42.3**	**73.9**	54.8	**51.1**

Prefetch CTA allocation ablation. Table 9 sweeps the number of persistent-kernel CTAs on NVIDIA H100 GPU with MiniCPM4.1-8B at 32K. The optimal count generally grows with batch size. Using 16 CTAs is best at small batches (B8–B16), while 32 (or 64) CTAs are preferred at larger batches. Our adaptive heuristic uses 16 CTAs when batch size < 32 and 32 CTAs otherwise, matching or staying within 4% of the best fixed configuration at every batch size. For the NVIDIA A100 GPU, a similar ablation yields an adaptive scheme of 16 CTAs when batch size < 64 and 32 CTAs otherwise.

Table 9: Prefetch CTA-count sweep on NVIDIA H100 GPU with MiniCPM4.1-8B at 32K context. Best throughput at each batch size, including ties, is **bolded**.

Method	B4	B8	B16	B32	B64	B128
SparDA (8 CTAs)	236.4	439.1	763.3	858.1	904.0	929.5
SparDA (16 CTAs)	235.7	**471.9**	**918.5**	1186.3	1255.9	1319.1
SparDA (32 CTAs)	233.5	471.5	813.1	**1205.2**	1328.0	**1358.2**
SparDA (64 CTAs)	**237.0**	**471.9**	737.3	1123.2	**1380.2**	1338.4
SparDA (adaptive)	235.7	**471.9**	**918.5**	**1205.2**	1328.0	**1358.2**

A100 throughput results. Tables 10 and 11 report prefill and decode throughput on an NVIDIA A100 GPU, providing a comparison with the H100 results in the main text.

Table 10: Prefill throughput (tok/s) on NVIDIA A100 GPU at batch size 4. Sequence length swept from 32K to 128K. †: no offload. Best value per column is **bolded**.

	MiniCPM4.1-8B				NOSA-8B			
Method	32K	64K	96K	128K	32K	64K	96K	128K
Dense†	**9531.5**	6836.1	5260.2	4282.9	**9576.2**	**6870.5**	5313.5	4316.0
Sparse†	8002.2	7310.1	6776.8	6310.3	5833.2	5364.6	5022.4	4713.5
Sparse	8052.3	7311.0	6765.9	6299.2	5839.7	5365.0	5021.9	4717.1
InfiniGen	7985.5	7268.6	6740.0	6296.3	5815.4	5339.0	5005.2	4711.5
SparDA	8555.9	**8191.3**	**7959.4**	**7756.8**	6097.4	5817.9	**5651.5**	**5477.5**

Table 11: Decode throughput (tok/s) on NVIDIA A100 GPU. Batch size (B) swept from 4 to 128. †: no offload. "–": OOM. Best value per column is **bolded**.

		MiniCPM4.1-8B						NOSA-8B					
Context	Method	B4	B8	B16	B32	B64	B128	B4	B8	B16	B32	B64	B128
32K	Dense†	160.9	219.7	268.8	–	–	–	**161.0**	218.6	266.8	–	–	–
	Sparse†	**171.4**	**348.6**	**688.7**	–	–	–	155.7	307.8	**630.3**	–	–	–
	Sparse	165.6	276.1	405.9	515.7	597.9	649.2	158.5	**317.8**	572.7	828.2	1097.9	1113.3
	InfiniGen	55.2	71.8	75.2	71.3	72.3	27.3	46.8	108.1	154.4	182.3	157.8	119.8
	SparDA	160.4	316.4	568.0	**649.2**	**668.9**	**691.9**	148.1	292.4	591.4	**988.6**	**1211.1**	**1218.1**
64K	Dense†	109.6	134.8	–	–	–	–	109.5	133.7	–	–	–	–
	Sparse†	**173.2**	**344.6**	–	–	–	–	157.4	**310.3**	–	–	–	–
	Sparse	147.3	236.7	345.6	443.9	516.1	556.6	**158.5**	284.6	479.8	704.8	911.8	861.3
	InfiniGen	54.0	67.5	66.9	63.3	27.2	–	63.3	100.1	133.5	138.1	62.5	–
	SparDA	160.3	316.3	**457.8**	**573.7**	**593.1**	**578.4**	146.9	292.8	**577.3**	**887.8**	**1042.6**	**961.3**
96K	Dense†	83.3	97.4	–	–	–	–	82.9	96.7	–	–	–	–
	Sparse†	157.8	301.6	–	–	–	–	**153.0**	294.8	–	–	–	–
	Sparse	127.6	208.7	310.7	402.5	467.0	**503.1**	140.7	253.5	424.0	629.9	814.2	**849.0**
	InfiniGen	50.8	62.0	57.5	53.2	–	–	61.8	97.0	124.7	122.6	–	–
	SparDA	**158.7**	**306.6**	**436.0**	**522.6**	**529.4**	–	146.1	**295.2**	**568.1**	**778.4**	**930.5**	–
128K	Dense†	67.2	–	–	–	–	–	66.6	75.3	–	–	–	–
	Sparse†	141.0	–	–	–	–	–	136.5	–	–	–	–	–
	Sparse	117.3	195.1	287.3	383.1	447.9	–	127.4	232.4	395.6	585.4	795.9	–
	InfiniGen	49.6	62.3	58.1	41.3	–	–	62.8	95.7	70.3	62.2	–	–
	SparDA	**160.2**	**303.3**	**424.4**	**485.5**	**501.4**	–	**148.4**	**298.2**	**526.3**	**691.3**	**876.9**	–

On prefill, SparDA achieves up to 1.23× speedup over Sparse and 1.81× over Dense on MiniCPM4.1-8B at 128K, and up to 1.16× over Sparse on NOSA-8B. These gains are consistent with the H100 results, confirming that the Forecast indexer's reduced selection cost transfers across GPU generations.

On decode, SparDA achieves up to 1.55× speedup over Sparse on MiniCPM4.1-8B (128K, B8) and up to 1.34× on NOSA-8B (96K, B16), slightly lower than on H100. The same trends hold: speedups peak at middle batch sizes and NOSA-8B shows lower gains due to its reduced KV fetch traffic. InfiniGen's CPU-side gather bottleneck is even more pronounced on A100, with throughput degrading sharply at larger batch sizes.

Decode speedup breakdown. Table 12 isolates the two sources of SparDA's decode speedup on NVIDIA H100 GPU at 128K sequence length: reduced block-selection cost and overlapped CPU-to-GPU prefetch. SparDA (no prefetch) uses the Forecast indexer but fetches missing KV blocks synchronously as in Sparse, without asynchronous overlap. Even without prefetch, this variant already outperforms Sparse at B4 on both models because the Forecast indexer cuts block-selection latency. At this small batch, the prefetch pipeline adds slight overhead that makes the full SparDA marginally slower than the no-prefetch variant. As batch size grows, layer execution scales sub-linearly while KV transfer volume grows linearly. Prefetch overlap therefore becomes increasingly beneficial. By B16 the full SparDA pulls ahead of the no-prefetch variant on both models, and by B64 it is roughly 40% faster. At B64, SparDA (no prefetch) drops below Sparse on both models because the block-selection savings diminish at large batches while the indexer projection cost remains, confirming that prefetch overlap is essential for SparDA's advantage at high batch sizes.

Table 12: Decode throughput (tok/s) breakdown on NVIDIA H100 GPU at 128K sequence length. Batch size (B) swept from 4 to 64. Best value per column is **bolded**.

	MiniCPM4.1-8B					NOSA-8B				
Method	B4	B8	B16	B32	B64	B4	B8	B16	B32	B64
Sparse	167.8	279.5	447.9	618.6	788.9	173.2	285.4	529.2	898.7	1298.3
SparDA (no prefetch)	**247.3**	379.1	505.9	608.2	696.9	**226.5**	**434.9**	653.0	884.2	1063.8
SparDA	240.2	**471.2**	**705.3**	**899.2**	**1000.1**	219.0	399.3	**735.0**	**1127.0**	**1463.3**

Per-task accuracy breakdowns. Tables 13–16 report per-task scores for each benchmark family, complementing the aggregated results in Table 1.

Table 13: Per-category HELMET scores. Best among sparse methods (excluding Dense) is **bolded**.

	MiniCPM4.1-8B (64K)				NOSA-8B (32K)			
Category	Dense	Sparse	InfiniGen	SparDA	Dense	Sparse	InfiniGen	SparDA
Recall	84.7	**67.8**	37.8	61.5	77.9	40.9	19.6	**45.6**
RAG	55.1	**54.7**	51.4	54.6	51.4	49.1	47.4	**49.4**
ICL	84.0	82.2	**82.3**	81.9	64.3	68.4	**69.1**	67.9
Citation	2.5	**3.5**	3.0	3.2	3.4	3.2	2.9	**3.9**
Rerank	12.5	14.7	13.5	**17.0**	32.7	22.3	15.9	**24.7**
LongQA	31.8	30.9	30.2	**32.1**	28.3	26.9	**27.4**	26.0
Summ	20.9	**18.2**	16.4	18.0	17.0	14.8	14.6	**16.5**
Average	41.7	**38.9**	33.5	38.3	39.3	32.2	28.1	**33.4**

Table 14: Per-task LongBench scores. Best among sparse methods (excluding Dense) is **bolded**.

	MiniCPM4.1-8B				NOSA-8B			
Task	Dense	Sparse	InfiniGen	SparDA	Dense	Sparse	InfiniGen	SparDA
2wikimqa	33.1	35.2	**35.4**	34.2	37.8	**36.8**	35.5	35.5
gov_report	28.5	**28.2**	27.8	28.0	32.8	33.3	32.9	**33.6**
hotpotqa	48.6	48.9	**49.6**	49.2	50.5	48.0	47.5	**48.2**
multifieldqa_en	53.3	53.8	53.5	**54.3**	50.5	51.4	51.4	**51.5**
musique	23.7	**25.4**	24.6	25.2	25.0	23.4	21.0	**24.9**
narrativeqa	20.7	20.2	22.5	**22.6**	23.1	**23.0**	22.0	21.8
passage_count	4.0	**3.5**	**3.5**	**3.5**	3.5	3.0	**4.0**	3.0
passage_retr.	100.0	**100.0**	**100.0**	**100.0**	89.0	**92.0**	87.0	90.5
qasper	42.0	**42.0**	41.8	41.9	34.8	**34.6**	33.9	34.3
qmsum	24.2	23.6	**24.0**	23.7	24.8	**25.0**	24.5	24.7
repobench-p	50.6	50.9	**51.2**	50.0	53.9	**54.1**	53.8	53.6
samsum	39.8	**40.8**	39.8	40.6	6.9	**6.1**	6.0	**6.1**
trec	76.0	**74.5**	**74.5**	**74.5**	74.5	74.5	75.0	**75.5**
triviaqa	83.0	**83.6**	**83.6**	**83.6**	88.2	**88.2**	87.7	**88.2**
Average	44.8	45.0	**45.1**	**45.1**	42.5	**42.4**	41.6	42.3

Table 15: Per-task RULER scores at the native maximum (64K for MiniCPM4.1-8B, 32K for NOSA-8B). Best among sparse methods (excluding Dense) is **bolded**.

	MiniCPM4.1-8B (64K)				NOSA-8B (32K)			
Task	Dense	Sparse	InfiniGen	SparDA	Dense	Sparse	InfiniGen	SparDA
niah_single_1	100.0	**100.0**	**100.0**	**100.0**	100.0	**100.0**	**100.0**	**100.0**
niah_single_2	100.0	**100.0**	**100.0**	**100.0**	100.0	**100.0**	**100.0**	**100.0**
niah_single_3	100.0	**100.0**	86.0	**100.0**	100.0	**100.0**	98.0	**100.0**
niah_multikey_1	98.0	94.0	96.0	**100.0**	100.0	86.0	70.0	**90.0**
niah_multikey_2	98.0	**80.0**	44.0	78.0	98.0	50.0	18.0	**52.0**
niah_multikey_3	98.0	**62.0**	28.0	56.0	90.0	20.0	8.0	**28.0**
niah_multivalue	94.5	95.5	90.5	**96.5**	94.0	92.0	85.0	**95.0**
niah_multiquery	99.0	97.5	87.5	**98.5**	95.0	90.0	71.5	**95.0**
vt	54.8	74.4	73.2	**74.8**	98.8	97.2	97.2	**98.4**
cwe	44.4	11.0	**23.2**	12.4	47.6	31.6	**37.8**	34.6
fwe	92.0	**94.0**	79.3	92.7	95.3	89.3	**90.7**	86.0
qa_1	80.0	64.0	44.0	**70.0**	56.0	38.0	32.0	**40.0**
qa_2	50.0	**44.0**	38.0	**44.0**	46.0	**44.0**	40.0	42.0
Average	85.3	78.2	68.4	**78.7**	86.2	72.2	65.2	**73.9**

Table 16: Per-dataset reasoning accuracy (GPT-5.2 judge, %). All AIME entries are eight-seed means ± standard deviations, as in Table 7; MATH-500 is single-run. *Average* is the mean of the three datasets, computed before rounding. Best among sparse methods (excluding Dense) is **bolded**.

Model	Dataset	Dense	Sparse	InfiniGen	SparDA
MiniCPM4.1-8B	MATH-500	96.8	97.6	**97.8**	97.4
	AIME 2024	84.6 ± 4.1	83.8 ± 4.9	82.9 ± 3.5	$\mathbf{84.2} \pm 4.6$
	AIME 2025	76.3 ± 4.2	72.1 ± 7.9	72.9 ± 3.9	$\mathbf{75.4} \pm 5.6$
	Average	85.9	84.5	84.5	**85.7**
NOSA-8B	MATH-500	68.2	72.2	**72.8**	71.6
	AIME 2024	55.8 ± 7.0	49.2 ± 10.4	54.6 ± 9.1	$\mathbf{56.7} \pm 5.6$
	AIME 2025	32.1 ± 5.5	30.0 ± 9.4	35.8 ± 7.2	$\mathbf{36.2} \pm 8.4$
	Average	52.0	50.5	54.4	**54.8**

D.1 Retrieval Failure Analysis

Diagnostic evaluation. We instrument decode-time block selection on MiniCPM4.1-8B at 64K context across all 13 RULER tasks, using 20 examples per task. This is a separate diagnostic evaluation from the 50-example-per-task results in Table 15. Sparse selects with the actual query and coarse compressed keys, while SparDA uses the Forecast. The reference is the finer attention-derived training target described in Section 4.2; for this analysis, its scores are pooled to 64-token blocks before selecting the reference top-k set.

Selection agreement and answer coverage. *Target recall* is the fraction of target-selected blocks retained by a selector. *Needle recall* is the fraction of answer-containing blocks retained. Table 17 reports target recall for Sparse and SparDA, and needle recall for the target and both selectors. These metrics distinguish agreement with the training reference from coverage of the answer locations.

Table 17: Retrieval diagnostics on MiniCPM4.1-8B at 64K, with 20 examples per RULER task. Task scores are percentages; recalls are proportions. The target is the fine attention-derived training reference, evaluated at the same block granularity as the two selectors. These diagnostic scores are reported separately from Table 15.

	Task score (%)			Target recall		Needle recall		
Task	Dense	Sparse	SparDA	Sparse	SparDA	Target	Sparse	SparDA
niah_single_1	100	100	100	0.68	0.49	0.89	0.84	0.86
niah_single_2	100	100	100	0.57	0.49	0.76	0.66	0.70
niah_single_3	100	100	100	0.61	0.51	0.68	0.64	0.64
niah_multikey_1	100	95	100	0.57	0.49	0.71	0.60	0.64
niah_multikey_2	100	85	85	0.57	0.48	0.30	0.26	0.23
niah_multikey_3	100	75	65	0.56	0.47	0.28	0.27	0.25
niah_multiquery	99	100	99	0.59	0.50	0.56	0.43	0.50
niah_multivalue	90	94	95	0.58	0.50	0.62	0.49	0.56
vt	52	69	66	0.68	0.54	0.83	0.68	0.73
qa_1	80	65	70	0.58	0.50	0.26	0.22	0.23
qa_2	40	40	40	0.56	0.48	0.29	0.26	0.29
cwe	41	11	11	0.55	0.47	0.08	0.07	0.07
fwe	92	93	93	0.52	0.44	0.18	0.14	0.20

Shared failures on harder retrieval tasks. Across multikey_1–3, SparDA's target recall changes little (0.49 to 0.47), while the target's own needle recall falls from 0.71 to 0.28. Sparse and SparDA show similar declines in needle recall, from 0.60 to 0.27 and from 0.64 to 0.25, respectively. Their task scores also fall, while Dense stays at 100. On cwe, even the target retains only 0.08 of the answer-containing blocks; Sparse and SparDA both retain 0.07 and score 11, versus 41 for Dense. These observations suggest a shared difficulty in retaining relevant blocks that is not captured by Forecast-to-target agreement alone.

Forecast-specific differences and interpretation. SparDA has lower target recall than Sparse on all 13 tasks, yet equal or higher needle recall on 11. Its two needle-recall deficits occur on multikey_2

and multikey_3 (0.03 and 0.02). Accuracy is tied on multikey_2, but SparDA scores 65 versus 75 on multikey_3, a difference of two answers in this 20-example evaluation. This leaves room for an additional Forecast-selection cost on the hardest multikey task. Needle recall is also not a complete predictor of task accuracy. On vt, SparDA retains more answer-containing blocks (0.73 versus 0.68) but scores lower (66 versus 69). The study identifies retrieval failure patterns; it does not isolate the causal contribution of Forecast error.

E Architectural Transfer to Token-Level DSA

GLM-5.1 adaptation. GLM-5.1 is a 754B mixture-of-experts model with native token-level DSA [35, 12]. We freeze the backbone and train only the Forecast projections at 64K sequence length, using KL distillation against the native DSA indexer's selection distribution. Thus, the teacher is the DSA indexer, rather than the full attention query used by the block-sparse backbones. The Forecast retains the native indexer's 32 scoring heads with head dimension 128. This indexer is already smaller than the model's 64-head attention module. Reducing the Forecast head count further degraded accuracy. We therefore do not obtain the prefill selection-compute savings of the compact GQA indexer. In the GLM-5.1 adaptation, Forecast predicts the next layer's token selection. The baseline uses the native DSA selector.

Evaluation settings. Both configurations run in SGLang [40] on four nodes with eight NVIDIA H100 GPUs each (32 GPUs total). Tensor parallelism is 8, pipeline parallelism is 4, and expert parallelism is 8 within the same per-node GPU groups. RULER, HELMET, and LongBench use the 64K setting. RULER evaluates 50 examples for each of its 13 tasks; LongBench includes the same 14 tasks as the 8B evaluation, and HELMET reports seven category averages. GPT-5.2 is used for HELMET metrics requiring an LLM judge and for the reasoning suite. Reasoning uses a maximum generation budget of 32,768 tokens, temperature 0.9, and top-p 0.95. MATH-500 is evaluated once; AIME 2024 and 2025 each use eight sampling seeds (1–8), shared between native DSA and SparDA, with one sampled answer per problem per seed. We report the mean and standard deviation of per-seed AIME accuracies, as in Appendix C. These GLM experiments evaluate accuracy.

Results. Tables 18–21 provide the detailed scores, complementing the benchmark-family averages in Table 3 in Section 5.2. The positive AIME mean differences are within the reported seed-to-seed variation and are not presented as statistically significant gains. Family and overall averages are computed before rounding; detailed scores are displayed to one decimal place.

Table 18: GLM-5.1 HELMET scores by category at 64K.

Category	Native DSA	SparDA
Recall	100.0	99.8
RAG	75.6	75.6
ICL	90.2	90.8
Citation	52.4	51.3
Rerank	72.3	69.7
LongQA	62.0	62.5
Summ	45.3	44.4
Average	71.1	70.6

Table 19: GLM-5.1 LongBench scores by task at the 64K setting.

Task	Native DSA	SparDA
2wikimqa	75.3	74.0
gov_report	31.6	31.6
hotpotqa	68.3	67.6
multifieldqa_en	54.6	54.3
musique	56.4	56.5
narrativeqa	34.0	34.2
passage_count	1.7	1.0
passage_retr.	100.0	100.0
qasper	49.6	49.8
qmsum	23.4	23.2
repobench-p	67.3	68.4
samsum	39.2	39.6
trec	82.5	82.0
triviaqa	93.5	93.4
Average	55.5	55.4

Table 20: GLM-5.1 RULER scores by task at 64K.

Task	Native DSA	SparDA
niah_single_1	100.0	100.0
niah_single_2	100.0	100.0
niah_single_3	100.0	100.0
niah_multikey_1	100.0	100.0
niah_multikey_2	100.0	100.0
niah_multikey_3	100.0	100.0
niah_multivalue	100.0	99.5
niah_multiquery	100.0	100.0
vt	100.0	100.0
cwe	100.0	100.0
fwe	98.0	98.0
qa_1	84.0	84.0
qa_2	72.0	72.0
Average	96.5	96.4

Table 21: GLM-5.1 reasoning accuracy (GPT-5.2 judge, %) with a 32K generation budget. AIME entries are eight-seed means $\pm$ standard deviations; MATH-500 is single-run. *Average* is the mean of the three datasets.

Dataset	Native DSA	SparDA
MATH-500	99.6	99.8
AIME 2024	98.8 ± 1.7	99.2 ± 1.5
AIME 2025	93.8 ± 3.8	96.2 ± 3.3
Average	97.4	98.4

F Broader Impact

SparDA is an inference-efficiency technique that reduces the cost of serving long-context language models. By lowering the hardware required for a given throughput target, it could make long-context LLM capabilities more accessible to researchers and practitioners with limited GPU budgets. The same cost reduction can also lower the barrier to generating harmful content at scale. We do not introduce new model capabilities or training data, so the misuse surface is inherited from the underlying models rather than created by this work. We therefore encourage practitioners to apply existing safety filters, monitoring, and deployment controls when serving models accelerated by SparDA or similar techniques.

G Licenses and Copyright for External Assets

Table 22 lists the external assets used in this work and their reported licenses or copyright information. For AIME 2024 and AIME 2025, MAA denotes the Mathematical Association of America, which holds copyright in the original competition problems; the benchmark names link to the dataset sources.

Table 22: License and copyright information for external assets used in this paper.

Asset	Type	License / copyright
MiniCPM4.1-8B	Model	Apache 2.0
NOSA-8B	Model	Apache 2.0
GLM-5.1	Model	MIT
ProLong-64K	Dataset	MIT
HELMET	Benchmark	MIT
LongBench	Benchmark	MIT
RULER	Benchmark	Apache 2.0
MATH-500	Benchmark	MIT
AIME 2024	Benchmark	MAA copyright
AIME 2025	Benchmark	MAA copyright
NOSI	Library	MIT
SGLang	Library	Apache 2.0
Triton	Library	MIT
PyTorch	Library	BSD-3-Clause

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state the model-side and runtime contributions and summarize the measured accuracy and throughput gains on the two 8B models and the accuracy-only transfer to GLM-5.1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 6 discusses the method's limitations and the scope of the evaluation.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theorems or formal proofs. Section 4 gives equations for the proposed inference architecture and runtime pipeline instead.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The architecture and training objective are described in Sections 4 and 4.2. All training hyperparameters, evaluation settings, and hardware specifications are provided in Section 5 and Appendices B–C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The source code is released at `https://github.com/NVlabs/SparDA`, linked in the abstract. The method, training objective, hyperparameters, and evaluation settings are described in Section 4 and Appendices B–C. The evaluated base models and benchmarks are public and cited, with license and copyright information in Table 22.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training details (optimizer, hyperparameters, data) are in Section 4.2 and Appendix B. Evaluation details (models, benchmarks, decoding strategy, hardware) are in Section 5 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: HELMET, LongBench, and RULER use deterministic decoding and are reported as single runs. For AIME 2024/2025, all four 8B configurations report the mean and standard deviation over eight sampling seeds (Appendix C); MATH-500 and the coarse-window ablation are single runs. Throughput numbers average two measured runs after one warmup run. We do not report confidence intervals or statistical significance tests.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 5 and Appendices B–C report GPU type and memory, CPU memory, CPU–GPU interconnect, training steps, GPU count and wall-clock time for indexer training, and the throughput measurement protocol. Section 6 and Appendix E describe the GLM-5.1 evaluation on four nodes of eight H100 GPUs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: To the authors' knowledge, the work conforms to the NeurIPS Code of Ethics. The paper is a systems and efficiency study on existing models and benchmark suites, with no human subjects or new data collection.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix F discusses both the positive impact (lowering hardware cost for long-context inference) and the negative risk (cheaper generation of harmful content), and notes that the misuse surface is inherited from the underlying models.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new public model, dataset, or service endpoint. It reports efficiency and accuracy measurements on existing long-context models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

Justification: External models, datasets, benchmarks, and libraries are identified in the paper. Table 22 in Appendix G reports their license or copyright information, including links to the AIME dataset sources and attribution of the original AIME problems to the Mathematical Association of America.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The SparDA source code is released at `https://github.com/NVlabs/SparDA`. The repository README provides installation, data preparation, training, and evaluation instructions. License information and third-party attributions are included alongside the code.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The work does not involve crowdsourcing or experiments with human participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve human subjects research or data collection from participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The paper modifies the inference architecture of existing LLMs but does not use LLMs as a non-standard component of the core method. GPT-5.2 is used as a standard evaluation judge for HELMET and reasoning benchmarks.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.