# DisSparse: Pipelined Top-$p$ Sparse Attention for Long-Context LLM Serving

**Nurlan Nazaraliyev, Elaheh Sadredini, Nael Abu-Ghazaleh**
Department of Computer Science and Engineering
University of California, Riverside
{nnaza008, elahehs, naelag}@ucr.edu

## Abstract

Long-context LLM inference is bottlenecked by the memory demands of the key-value (KV) cache, whose size scales linearly with context length. Dynamic sparse attention reduces this bottleneck by fetching and attending only to a critical subset of tokens. The dominant primitive in dynamic sparse attention research and deployment is fixed-budget top-$k$ selection, which is favored because its deterministic block count aligns with the static memory allocation that systems like vLLM rely on. Top-$k$ uses a fixed budget regardless of how attention mass is distributed across heads, layers, and decode steps, leading to either wasted bandwidth on heads with peaky attention or dropped context on heads with diffuse attention. Top-$p$ (nucleus) selection adapts naturally to this variation but has seen limited adoption in production serving systems due to two system-level challenges: variable block counts that break deterministic block allocation and computationally expensive selection kernels that exceed the cost of attention itself. We present DisSparse, a serving system that integrates top-$p$ sparse attention into vLLM end-to-end. DisSparse decouples block selection from the scheduler's critical path, hides selection latency behind layer computation via a two-stream pipeline, and introduces a single-pass histogram-based top-$p$ kernel. We further observe *intra-block sparsity*, the inflation of selection budgets caused by mixing high- and low-importance tokens within fixed-size KV cache blocks, and address it with a token relocation kernel guided by accumulated prefill attention scores. Across long- and medium-context benchmarks on multiple models, DisSparse achieves 1.17$\times$ to 1.80$\times$ speedup in sparsity analysis (block selection) over state-of-the-art top-$p$ methods with negligible accuracy loss and supports up to 8$\times$ larger maximum batch sizes than dense vLLM under the same memory budget. Code is available at `https://github.com/nnurlan008/DisSparse`

## 1 Introduction

Large language models (LLMs) now support context windows spanning hundreds of thousands of tokens, with frontier models reaching one million or more, enabling applications such as document summarization [12, 11], multi-turn dialogue [16], repository-scale code generation, agentic workflows [18], and long-context reasoning [13]. However, these advances intensify a primary bottleneck: the key-value (KV) cache that stores the conversation history must reside in fast accelerator memory, and its size grows linearly with context length. At long contexts, the KV cache dominates both memory capacity and bandwidth [68], limiting the number of concurrent requests a serving system can sustain and the latency it can deliver.

Two complementary lines of work address this bottleneck. *KV cache compression* reduces the per-token footprint via quantization [21, 22, 20] or low-rank approximation [73, 74]. *Sparse attention* reduces the number of tokens accessed per query, either by evicting tokens from the cache [23] or by dynamically selecting subsets [7, 1]. The two axes are orthogonal: compression shrinks each token's representation while reading every token; sparsity skips tokens entirely. Combined with offloading, sparsity also reduces the GPU-resident footprint.

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

Sparse attention divides into *static* approaches that commit to a fixed sparsity pattern offline [24, 25] and *dynamic* approaches that select important tokens per query at runtime [26, 7, 1]. Static patterns are simple to implement but cannot adapt to attention distributions that shift across layers, heads, and decoding steps, nor to the fact that token importance itself evolves during generation [36]. Dynamic methods address this fundamental limitation, but introduce system-level challenges of their own [4, 55].

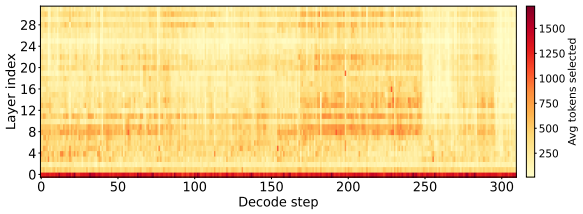



Figure 1: Llama-3.1-8B-Instruct [47] on NarrativeQA from LongBench [12]: critical token counts per layer per decode step vary substantially, showing why fixed-$k$ budgets are mismatched. Results on other models are in Appendix A.

The dominant dynamic-sparsity primitive is *top-$k$*: select a fixed number of highest-scoring blocks per head [27, 3, 56, 1, 58]. Top-$k$ is attractive because $k$ is a compile-time constant; memory allocation, kernel launch parameters, and scheduling decisions can all be determined before the query arrives [1, 5]. However, a fixed $k$ is mismatched to attention distributions that vary widely across layers, heads, and generation steps (Figure 1). A $k$ tuned for the densest heads wastes bandwidth on sparse heads; a $k$ tuned for the typical heads drops critical context on dense ones. Previous studies have measured these failures: fixed-budget top-$k$ violates target attention mass in over 20% of heads [62] and degrades on tasks with diffuse attention [26].

*Top-$p$ (nucleus) selection*, retaining exactly the blocks whose cumulative probability mass reaches threshold $p$, adapts naturally to this variation. Despite its appeal, top-$p$ has seen little adoption in production serving systems due to two challenges: **(1)** existing top-$p$ implementations are computationally expensive, locating the exact threshold via multi-pass search [7] or accumulating partial attention until $p$ is reached [46]; and **(2)** the number of selected blocks is unknown until selection completes, complicating the deterministic memory allocation that schedulers like vLLM rely on. Both increase selection cost, as Figure 3 shows, top-$p$ selection [7] starts dominating layer execution time at batch sizes $\geq 16$.

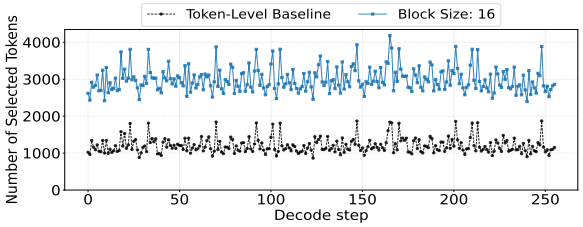



Figure 2: Llama-3.1-8B-Instruct [47], Layer 0 on LongBenchV2 [11]. Block-level top-$p$ selects $\sim 2\times$ more tokens than a token-level oracle to reach $p = 0.95$, motivating the relocation kernel.

In this paper, we present DisSparse, a system that integrates top-$p$ sparse attention into vLLM [5] end-to-end. DisSparse addresses these challenges via co-designed components. *For challenge (1)*, we introduce a histogram-based kernel that computes an approximate threshold in a single pass over $N$ tokens using $B$ histogram bins, replacing multi-pass binary search. *For challenge (2)*, our key insight is that top-$p$ selection can be *decoupled from the scheduler's critical path*: DisSparse delegates retrieval of already-computed KV blocks to the model executor, leaving only new-block allocation in the scheduler. To overlap the resulting selection latency with computation, we organize each decoding step into two CUDA streams: a *selection stream* that runs top-$p$ analysis on an INT4 KV cache replica [7] for layer $\ell{+}1$ and prefetches the selected blocks, and a *compute stream* that executes self-attention and the FFN for layer $\ell$, building on high cosine similarity of consecutive layers' hidden states [53].

Beyond these two challenges, we observe an additional inefficiency specific to block-paged KV caches: tokens within a block often have highly skewed importance, so block-level top-$p$ must select more blocks than a token-level oracle to reach the same $p$ (Figure 2). We call this *intra-block sparsity* and introduce a relocation algorithm that re-clusters tokens post-prefill by their accumulated prefill attention scores. While no single relocation can perfectly predict importance at every future decode step, tokens with high cumulative prefill scores tend to remain important during generation [43, 44]; grouping them into shared blocks reduces the number of selected blocks by up to 14%.

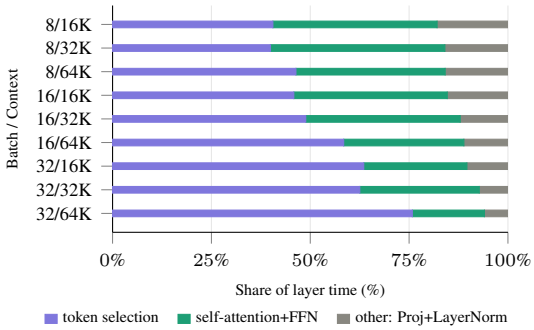



Figure 3: Per-layer time breakdown for Llama-3.1-8B-Instruct [47]. Selection grows from $\sim$40% to $\sim$76% of layer time as batch and sequence length scale, dominating attention and FFN combined. Measurements are based on individual kernels.

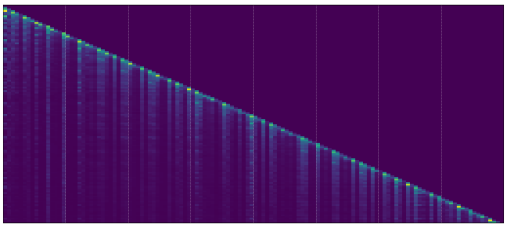


Figure 4: Layer 0 attention (Llama-3.1-8B-Instruct, LongBenchV2); white dashed lines mark 16-token blocks. Attention concentrates on a few tokens per block, yet block-level top-$p$ must load the whole block, motivating intra-block relocation.

We evaluate DisSparse on long-context [12, 11] and medium-context [59–61] benchmarks across Llama 3.1-8B-Instruct [47], LLaMA2-7B-Chat [65], Longchat-7B-v1.5-32k [50], Qwen2.5-7B-Instruct [17], and Phi-3-mini-128k-instruct [49]. DisSparse achieves 1.17$\times$ to 1.80$\times$ speedup over Twilight [7], the state-of-the-art top-$p$ method, with negligible accuracy loss. At the system level, DisSparse's integration into vLLM [5] increases the maximum sustainable batch size by up to 8$\times$ over dense vLLM under the same GPU memory budget.

Our contributions are:

1. A **pipelined top-$p$ sparse attention system** that decouples block selection from the scheduler's critical path and overlaps selection and prefetch with layer computation via a two-stream design, integrating into vLLM [5] with minimal scheduler modifications (§4.1).
2. A **histogram-based approximate top-$p$ kernel** that replaces multi-pass binary search [7] or progressive accumulation [46] with a single $O(N)$ pass using $O(B)$ registers (§4.2).
3. A **token relocation kernel** that re-clusters KV pairs by accumulated prefill attention scores, reducing the number of blocks selected by top-$p$ by up to 14% (§4.2).
4. A **comprehensive evaluation** on long- and medium-context benchmarks across multiple models, demonstrating up to 1.80$\times$ selection speedup over Twilight [7] with negligible accuracy degradation and up to 8$\times$ batch-size increase over dense vLLM (§5).

## 2 Background and Related Work

**Sparse Attention.** Attention distributions in LLMs are inherently skewed: a small fraction of tokens dominates the attention mass [34, 36], and the majority can be skipped during decoding with negligible accuracy loss [35]. *Static* approaches (e.g., sliding window, dilated attention) apply fixed token-dropping patterns but typically require training or fine-tuning with the fixed pattern [25]. *Dynamic* sparse attention (DSA) instead adapts on the fly to input context. Within DSA, *training-aware* methods modify the architecture and require continued pre-training [37], while *training-free* methods [1–3] operate at inference time on off-the-shelf models. The dominant primitive in training-free DSA is *top-$k$* [1, 3, 56, 58], but it cannot adapt to the wide variation in attention distributions across layers, heads, and decoding steps, leading to accuracy degradation on tasks with diffuse attention patterns [26, 62].

**Top-$p$ Sparse Attention.** *Top-$p$ (nucleus)* selection retains the minimal set of tokens whose cumulative attention mass exceeds threshold $p$. Twilight [7] first selects candidate pages via a fixed top-$k$ budget, then applies token-level top-$p$ pruning through binary search; the multi-pass search incurs significant selection overhead and requires multiple separately launched kernels (GEMV, softmax, and top-$p$ search). Progressive Sparse Attention [46] processes blocks in ranked order and accumulates mass until the threshold is met, but its iterative microbatch execution incurs substantial CPU–GPU

synchronization overhead. Double-P [62], concurrent work, performs hierarchical top-$p$ estimation via k-means centroids followed by token-level refinement. These methods advance the algorithmic design of top-$p$ but do not address the systems-level challenges of integrating variable-count selection into production serving frameworks; DisSparse fills this gap by co-designing top-$p$ with the scheduler, memory manager, and execution pipeline.

**KV Cache Compression.** Quantization reduces the per-token KV cache footprint, with fixed-precision [41, 42] and importance-adaptive [21] variants. A related but distinct line, eviction, permanently removes unimportant tokens: H2O [34] and Scissorhands [43] drop tokens with low accumulated attention, StreamingLLM [35] retains only initial tokens plus a sliding window, and SnapKV [44] compresses during prefill. While effective at reducing capacity, permanent eviction can destroy the context needed for long-range retrieval. Both are orthogonal to DisSparse and can be composed with our sparse attention to yield cumulative gains.

**Serving Systems and Asynchronous KV Management.** Modern serving frameworks [5, 6, 10, 9] manage the KV cache at block granularity via PagedAttention [5], which assumes the per-request block count is known at scheduling time, an assumption top-$p$ violates. These systems support either *hybrid* (colocated) or *disaggregated* [10, 9] prefill/decode; DisSparse adopts disaggregation to isolate decode-stage improvements but is compatible with both. To overlap data movement with computation, InfiniGen [53] prefetches KV blocks speculatively by exploiting the high cosine similarity between hidden states at consecutive layers, an observation consistent with broader evidence of consecutive layer representations' similarity in LLMs [52]. DisSparse builds on this insight: a two-stream pipeline runs top-$p$ analysis for layer $\ell{+}1$ alongside layer $\ell$'s attention and FFN, hiding selection and prefetch latency behind useful computation.

## 3 The Systems Challenge of Dynamic Top-$p$ Sparsity

While dynamic sparse attention (DSA) offers a promising path to reducing KV cache memory bandwidth during autoregressive decoding, integrating it into production serving frameworks introduces substantial systems-level friction [55]. To date, system-level efforts have largely converged on top-$k$ selection [1, 56] because its fixed budget aligns with the deterministic memory allocation in modern schedulers.

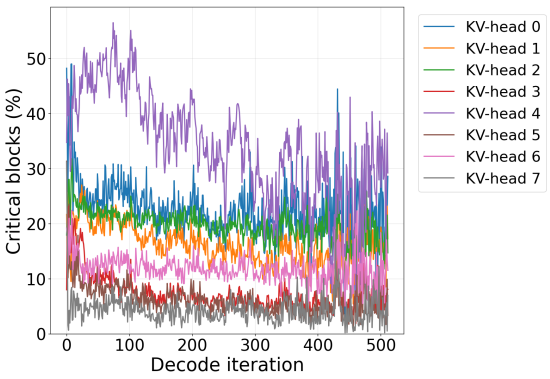



Figure 5: Percentage of selected critical blocks per KV head for layer 4 in Llama-3.1-8b-Instruct on LongBenchV2 samples. Results on other models are in Appendix A.

However, top-$k$'s fixed selection count is mismatched to the attention dynamics of modern LLMs. Figure 5 shows that attention distributions vary substantially across KV heads within a decode step and across steps for the same head. Prior work attributes this to head-level specialization: heads are either *retrieval* (gathering information from the full context) or *local* (focusing on nearby tokens) [14, 15]. A static $k$ over-selects on focused heads, wasting bandwidth on irrelevant blocks, and under-selects on dispersed ones, discarding blocks that contribute to output quality.

Top-$p$ (nucleus) selection can address this mismatch by adapting the budget per layer and head [57, 7], but faces the following challenges:

**Challenge 1: Cost of Relative Score Computation.** Top-$k$ can rank blocks from compact metadata such as element-wise bounds [1] or query-conditioned approximations [27], never touching the full cache. Top-$p$, by contrast, must compute and accumulate the actual attention scores across all cached tokens to locate where the cumulative threshold is met. Performing this on the full-precision (FP16) cache would require loading the data that sparse attention is designed to skip, negating the bandwidth reduction entirely. A practical alternative is to maintain a lightweight, quantized (INT4) replica of the KV cache dedicated to score computation. However, this introduces additional memory overhead and

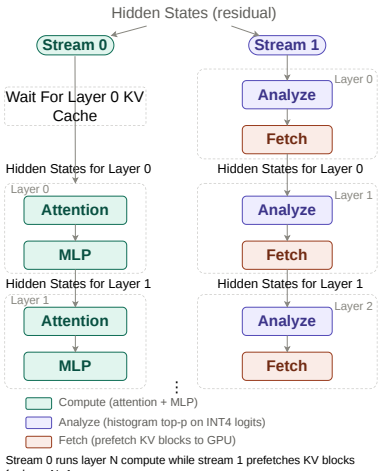



Figure 6: Pipelined Execution

**Algorithm 1**. Paged INT4 Scoring + Top-$p$

**procedure** TOPPSELECT($Q$, $K_{\text{INT4}}$, $P_{\text{tgt}}$, $k$, $B$)
**Phase 1 (scoring kernel): Per-block LSE** (context split across thread blocks)
  $m \leftarrow -\infty,\ l \leftarrow 0$
  **for** each tile $j$ of $K_{INT4}$-cache **do**
    $S_j \leftarrow (QK_j^\top) \cdot s$ ▷ INT4 dequant, optional ALIBI/softcap
    $V_j \leftarrow \max(S_j) + \log_2 \sum \exp_2(S_j - \max(S_j))$
    Update streaming $(m, l)$; store $V_j$ to block-LSE buffer
  **end for**
**Phase 2 (top-$p$ kernel): Top-$k$, then threshold $\tau$ via fast-path or histogram**
  $L \leftarrow m + \log_2 l$ ▷ global LSE
  $T \leftarrow \text{TopK}(\{V_j\}_j,\ k)$ ▷ $k$ rounds of block-parallel argmax
  $P_{\text{top}} \leftarrow \sum \exp_2(T - L),\ \hat{V}_{\min} \leftarrow \min(T) - L$
  **if** $P_{\text{top}} \geq P_{\text{tgt}}$ **then**
    $\tau \leftarrow \hat{V}_{\min} - \epsilon$ ▷ Fast path
  **else** ▷ Histogram path
    $P_{\text{above}} \leftarrow \sum_{j:\, V_j - L > \hat{V}_{\min}} \exp_2(V_j - L)$
    Build size-$B$ histogram $H$ of $\exp_2(V_j - L)$ for $V_j - L \in [\hat{V}_{\min} - W,\ \hat{V}_{\min}]$
    $b \leftarrow \max\{i : \text{RevCumSum}(H)_i \geq P_{\text{tgt}} - P_{\text{above}}\}$
    $\tau \leftarrow (\hat{V}_{\min} - W) + (b/B) \cdot W$ ▷ Slow path
  **end if**
  **return** blocks $j$ with $V_j - L \geq \tau$ (plus sink and local-window blocks)
**end procedure**

requires careful pipelining to avoid serializing the score computation with the main attention kernel (§4.1).

**Challenge 2: Non-Deterministic Memory Demand.** State-of-the-art frameworks such as vLLM rely on deterministic resource provisioning: the scheduler pre-allocates KV cache blocks from a shared free pool before execution, assuming the per-request block count is known at scheduling time. Top-$p$ violates this assumption: the number of selected blocks is determined only after computing attention scores and varies sharply across layers and heads within a single decode step (Figure 1). The scheduler must then either over-provision conservatively, wasting memory and shrinking batch sizes, or defer allocation to the model runner at runtime, breaking the separation between scheduling and execution. Neither is acceptable without a principled change to the allocation interface.

Beyond these deployment challenges, block-level granularity introduces an additional inefficiency. Modern inference engines manage the KV cache in blocks of 16 (32, 64, etc.) tokens via PagedAttention [5] to avoid memory fragmentation. Because attention scores within a block are often highly skewed, a single critical token forces retrieval of its entire enclosing block (Figure 4), inflating the block count needed to reach threshold $p$ well beyond the count of individually critical tokens. Figure 2 quantifies this gap: block-level top-$p$ requires $\sim 2\times$ more tokens than a token-level oracle for layer 0 of Llama-3.1-8b-Instruct on a LongBenchV2 [11] sample. The effect is worst under focused attention distributions, precisely the regime where top-$p$ should yield the greatest savings.

These two challenges are interconnected: the variable block count from top-$p$ (Challenge 2) compounds the cost of the score computation that produces it (Challenge 1), and intra-block sparsity further amplifies the bandwidth pressure both create. Addressing them in isolation yields partial solutions; effective deployment requires co-designing the selection algorithm with the memory management substrate.

# 4 DisSparse

## 4.1 System Architecture and Design

DisSparse uses a disaggregated serving architecture [66, 10], executing prefill and decode on separate devices to isolate decode-stage optimizations from prefill interference. We observe that, on long-context inputs (16K average sequence length), decode accounts for $\sim$78% of total inference time on a 2-GPU disaggregated vLLM setup, and this fraction grows with sequence length.

DisSparse maintains a *metadata cache*, an INT4 replica of the main FP16 key cache ($8\times$ smaller), used exclusively by the sparse analyzer for top-$p$ block selection. After the prefill engine generates the first token and its KV cache, the metadata cache is transferred to the decode engine via asynchronous

NCCL collectives [45]. During each decode step, the executor concurrently fetches selected FP16 KV blocks on the transfer stream while running the forward pass on the compute stream.

**Hybrid block allocation.** Standard vLLM uses a single scheduler-owned block allocator for newly generated tokens. Sparse attention with offloading introduces a second pattern: allocations for already-computed remote blocks fetched per step by top-$p$. The scheduler cannot model these efficiently as their count and identity are determined at execution time, not scheduling time. DisSparse therefore splits allocation between the scheduler and the device-side executor (Figure 7): the scheduler allocates blocks for newly generated tokens and the metadata cache, while the executor allocates blocks for transient sparse fetches. The two allocators synchronize at each step's end; the scheduler retains lifecycle responsibility (memory release, prefix caching). When the executor's free pool is insufficient, affected requests stall via a per-batch boolean mask without changing the batch configuration.

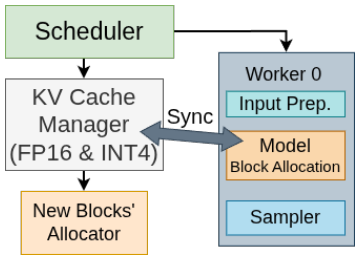



Figure 7: vLLM System Overview. DisSparse adds the new INT4 cache to the KV cache manager and synchronization unit between the workers (model executor) and the KV cache manager (scheduler).

**Computation-communication overlap.** Computing top-$p$ on the INT4 replica solves Challenge 1's bandwidth problem but introduces a new requirement: selection for layer $\ell$ must complete before its attention can fetch the chosen blocks, or the forward pass stalls. DisSparse avoids this by exploiting prior observations [53, 52] that consecutive layers' hidden states are highly similar; across our evaluated models, the cosine similarity ranges from 0.87 to 0.95[1], so the previous layer's hidden state reliably predicts the next layer's selection. Our two-stream pipeline (Figure 6) runs one layer ahead of compute. The *selection stream* runs top-$p$, fetches remote blocks, and allocates local slots for layer $\ell{+}1$ using a query computed from layer $\ell$'s hidden state through layer $\ell{+}1$'s $W_Q$. The *compute stream* runs layer $\ell$'s forward pass: self-attention over its previously selected blocks plus the surrounding projections, normalizations, FFN, etc.

### 4.2 Kernels Implementation

**INT4 Scoring and Top-$p$ Kernels.** DisSparse addresses Challenge 1 with two separate kernels: an INT4 GEMV kernel that scores blocks directly on the INT4 cache, and a histogram top-$p$ kernel that selects blocks from those scores in a single pass, replacing the multi-pass binary search of prior top-$p$ implementations. Both kernels run on the selection stream (§4.1); a condensed version is shown in Algorithm 1 (full version in Appendix D). The scoring kernel extends the tiling and online-softmax patterns of FlashAttention-2 [54], with grid axes indexed by query and KV head, and the top-$p$ kernel identifies critical KV cache blocks via a histogram approximation of top-$p$ over per-block log-sum-exp (LSE) scores.

**Algorithm 2** Importance-Guided Token Relocation

**Require:** KV caches $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{H_{kv} \times L \times d}$; prefill query/key $\mathbf{Q} \in \mathbb{R}^{H_q \times L \times d}, \hat{\mathbf{K}} \in \mathbb{R}^{H_{kv} \times L \times d}$; bands $R$
**Ensure:** Caches with important tokens concentrated in early blocks
**Phase 1: Accumulate prefill attention scores**
$\mathbf{A} \leftarrow \mathbf{Q}\hat{\mathbf{K}}^\top / \sqrt{d}$ ▷ Scaled dot-product with causal masking
$\mathbf{S}_h \leftarrow \sum_{t=1}^{L} \mathbf{A}_{h,t,:} \quad \forall\, h \in [H_q]$ ▷ Sum over query positions
**Phase 2: Aggregate importance per KV head**
**for** $h = 1$ **to** $H_{kv}$ **do**
  $\mathbf{s}_h \leftarrow \sum_{g=1}^{G} \mathbf{S}_{(h-1)G+g}$ ▷ Pool $G = H_q / H_{kv}$ grouped heads
**end for**
**Phase 3: Rank and partition into importance bands**
**for** $h = 1$ **to** $H_{kv}$ **do**
  $\boldsymbol{\pi}_h \leftarrow \mathrm{argsort}(\mathbf{s}_h,\ \texttt{descending})$ ▷ Rank $L$ tokens
  Partition top $R \cdot d_{\text{band}}$ into $R$ contiguous bands of size $d_{\text{band}}$; leave the tail in a single trailing band
  **for** each importance band $r = 1, \ldots, R$ **do**
    Sort indices within band $r$ in *ascending* order
  **end for**
  Sort indices in the tail band in *ascending* order
**end for**
**Phase 4: Reorder caches**
**for** $h = 1$ **to** $H_{kv}$ **do**
  $\mathbf{K}_h \leftarrow \mathrm{gather}(\mathbf{K}_h,\ \boldsymbol{\pi}_h)$; $\mathbf{V}_h \leftarrow \mathrm{gather}(\mathbf{V}_h,\ \boldsymbol{\pi}_h)$
**end for**

In Phase 1, the scoring kernel streams over $K_{\text{INT4}}$-tiles, computes raw attention logits $QK_{\text{INT4}}^\top$, and aggregates them into per-block LSE values (line 5), avoiding explicit per-token softmax. In parallel,

[1] 0.878 ($\pm$0.002) for Llama-2-7B-Chat [65] at 4K, 0.891 ($\pm$0.001) for Longchat-7B-v1.5-32k [50], 0.868 for Llama-3.1-8B-Instruct [47], 0.930 ($\pm$0.001) for Phi-3-mini-128k-instruct [49], and 0.951 ($\pm$0.001) for Qwen3-32B [64] at 32K.

it maintains a streaming buffer of the $k$ largest block LSEs (we use $k = 8$). In Phase 2, the top-$p$ kernel checks whether these Top-$k$ blocks already cover the target mass $P_{\text{tgt}}$. If so, the fast path sets $\tau$ just below $\hat{V}_{\min}$ (line 12), selecting the Top-$k$ blocks. Otherwise, the slow path constructs a $B = 64$-bin histogram of block probabilities over the log-probability window $[\hat{V}_{\min} - W, \hat{V}_{\min}]$ ($W = 8$, covering a $2^8\times$ probability range below the Top-$k$ boundary). The threshold $\tau$ is the lower edge of the highest bucket whose suffix mass covers the residual target $P_{\text{tgt}} - P_{\text{above}}$, where $P_{\text{above}}$ is the mass of blocks scoring above the Top-$k$ boundary, found via reverse cumulative summation. A final phase (in the full algorithm, Appendix D) atomically allocates the selected blocks from a free pool and returns their physical IDs ($-1$ for skipped blocks, compatible with the paged attention block table format). Throughout the paper, especially in evaluation section 5, rows marked *(token)* use a token-level variant of both kernels (Algorithm 3, Appendix D), with per-token scores instead of block LSEs and no Top-$k$ fast path or forced sink and local-window tokens.

Table 1 compares this approximation against an FP16 ground truth (exact sort-and-cumsum over unquantized softmax probabilities) on LongBenchV2 [11]. The kernel achieves $\geq 97\%$ recall across all thresholds while matching or slightly exceeding FP16 mass coverage, confirming INT4 + histogram selection as a high-fidelity drop-in replacement for exact FP16 top-$p$.

**Token Relocation Guided by Accumulated Prefill Attention.** Algorithm 2 details the token relocation procedure for mitigating *intra-block sparsity*. After prefill, tokens reside in chronological order in the KV cache. For block-sparse attention during decode, this ordering is suboptimal: tokens with high attention scores, those critical for reaching threshold $p$, may be scattered across many blocks, forcing top-$p$ to load far more blocks than necessary (Figure 2). We address this by physically rearranging KV cache entries after prefill, placing the most attended tokens into the earliest blocks so that top-$p$ reaches the same $p$ with fewer blocks. This is safe because RoPE [70] is applied to keys before caching, so positional information is already encoded in the keys after being cached. Importance is measured by the total attention each token receives during prefill (line 2), summed across the $G = H_q/H_{\text{kv}}$ query heads within each GQA group (line 4). Relocation is performed independently per KV head, since heads exhibit different attention patterns (Figure 5).

Table 1: Selection difference between DisSparse applied on INT4 vs. exact top-$p$ selection on FP16 with PyTorch [67] on LongBenchV2 samples. Mass: average cumulative probability of selected blocks.

| | Blocks | | | | Mass | |
|---|---|---|---|---|---|---|
| Threshold | FP16 | INT4 | Extra | Recall | FP16 | INT4 |
| $p = 0.8$ | 66.9 | 72.3 | +5.3 | 0.9754 | 0.8346 | 0.8612 |
| $p = 0.85$ | 89.1 | 94.9 | +5.8 | 0.9743 | 0.8737 | 0.8906 |
| $p = 0.9$ | 123.3 | 129.2 | +5.8 | 0.9723 | 0.9140 | 0.9227 |
| $p = 0.95$ | 187.4 | 193.5 | +6.0 | 0.9731 | 0.9559 | 0.9590 |

Naive sorting by accumulated attention destroys the causal locality that attention relies on, eliminating the benefit. Instead, we use *banded sorting* (Algorithm 2): tokens are ranked globally by attention scores, then partitioned into $R$ contiguous importance bands (band 1 most important, band $R$ least). Within each band, tokens are re-sorted by their original position to restore chronological order. The K and V caches are then permuted accordingly via gather operations per head. The result: early blocks concentrate the most attended tokens while preserving sequential structure within each band, reducing the number of blocks selected by top-$p$ by up to 14% (§5). Relocation runs once per request after prefill completes, with negligible cost relative to prefill itself.

## 5 Evaluation

We evaluate DisSparse on 6 different models: Llama 3.1-8B-Instruct[47], LLaMA2-7B-Chat[65], Longchat-7B-v1.5-32k[50], Qwen2.5-7B-Instruct [17], Qwen3-32B [64], and Phi-3-mini-128k-instruct[49] based on 2 parts: (1) Accuracy and kernel performance (§5.1) on NVIDIA A100, RTX 6000 Ada, and RTX PRO 6000 Blackwell GPUs; (2) System performance (§5.2), on different benchmarks using 2 A100-40GBs, one for prefill and another

Table 2: Average scores on 12 tasks from LongBench. Budget is the average number of selected tokens per layer/head. Detailed per-dataset results are in Table 5 (Appendix B). DS results are from [7, 51].

| | LongChat-7B | | LLaMA-3.1-8B | |
|---|---|---|---|---|
| **Method** | **Budget** | **Avg.** | **Budget** | **Avg.** |
| Full Attention | 32k | 36.10 | 32k | 52.01 |
| Quest+Twi | 137 | 37.33 +3.4% | 489 | 52.00 -0.0% |
| DS+Twi | 126 | 37.98 +5.2% | 446 | 51.73 -0.5% |
| Quest+DisSparse | 141 | 37.66 +4.3% | 520 | 52.10 +0.2% |
| DisSparse | 170 | **37.79** +4.7% | 610 | **52.20** +0.4% |

for decode engines. Benchmarks mainly include long-context understanding (LongBench [11, 12]), retrieval (RULER [71]), medium-context datasets: GSM8K [59], COQA[60] and PG-19's perplexity test[61].

### 5.1 Accuracy and Kernel Performance.

**Benchmarks**. For this experiment, we evaluate DisSparse for its accuracy and kernel performance. The baselines include different SOTA sparse-attention algorithms, such as Quest[1], DS[51], and Twilight[7], and full/dense attention. Following the baselines, we do not apply any sparse methods to the first two layers to ensure a fair comparison. For DS, we use the optimized configurations tuned for each model provided by its official repository. The threshold $p$ of Twilight and DisSparse is set to 0.95 for LLaMA-2/3, and 0.85 for Longchat [7] to align with other works. For all baselines, we follow their official repository.

**Accuracy Results on LongBench.** Table 5 summarizes the accuracy result of Twilight[7] integrated with Quest[1] and DS[51], dense attention, and DisSparse on 12 datasets from LongBench dataset[12]. Across both models, DisSparse matches or exceeds dense attention accuracy. On LongChat-7B-v1.5-32k, DisSparse improves by up to +4.7% over the dense baseline at a budget of only 170 tokens, likely because adaptive top-$p$ selection filters low-attention noise that contributes to output degradation. On LLaMA-3.1-8B-Instruct, DisSparse, and the baselines all land within ±0.5% of the dense baseline, within the natural variance of LongBench metrics. The key result is that DisSparse achieves this with substantially smaller selection budgets than dense attention requires, similar to Twilight baselines. DisSparse also outperforms Ada-KV [72] at similar budgets (Table 12).

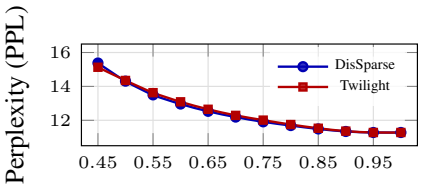



Figure 8: Perplexity on PG-19 vs. top-$p$ threshold for LLaMA-3.1-8B.

**Accuracy Results on medium-length context.** Table 3 reports accuracy on GSM8K [59], COQA [60], and PG-19 perplexity [61]. DisSparse matches Twilight and dense attention to within about 2% on both models across all metrics, while Quest-128 (fixed top-$k$) collapses on GSM8K (0.40 vs. dense 0.78) and PG-19 (PPL 79.23 vs. 11.28) on LLaMA-3.1, confirming that fixed-budget top-$k$ fails on diffuse attention while top-$p$ adapts. DisSparse's selection budget is within 12% of Twilight's (Table 3, italic rows). Figure 8 sweeps top-$p$ from 0.45 to 1.0 on LLaMA-3.1: DisSparse tracks Twilight across the range (gap $\leq 0.25$ PPL), and both converge to within 0.05 PPL of dense at $p \geq 0.95$. To match Twilight, DisSparse runs at token-level granularity for this experiment.

Table 3: Results for GSM8K, COQA, PG-19 (length 4K for LLaMA-2, 16K for LLaMA-3.1). Token Budget: Twilight/DisSparse.

| | GSM8K(flexible/strict)↑ | COQA(em/f1)↑ | PG-19 PPL↓ |
|---|---|---|---|
| LLaMA-2-7B-Chat | | | |
| Full (dense) | 0.227/0.225 | 0.606/0.761 | **10.85** |
| Quest-128 | 0.052/0.051 | 0.571/0.744 | 27.47 |
| Twilight | **0.225/0.225** | **0.599/0.759** | 11.09 |
| DisSparse | 0.223/0.221 | 0.597/0.758 | 11.07 |
| Token Budget | *105/117* | *112/125* | *415/452* |
| LLaMA-3.1-8B-Instruct | | | |
| Full (dense) | 0.776/0.713 | 0.642/0.784 | **11.28** |
| Quest-128 | 0.401/0.384 | 0.611/0.760 | 79.23 |
| Twilight | **0.785/0.737** | 0.639/0.781 | 11.28 |
| DisSparse | 0.779/0.726 | **0.643/0.784** | 11.28 |
| Token Budget | *89/97* | *93/103* | *1187/1230* |

**Accuracy Results on Retrieval.** A sparse method can fail on retrieval by dropping a rare but critical token. We therefore evaluate 11 RULER [71] subtasks on Llama-3.1-8B-Instruct at 16K and 32K. All sparse configurations stay within 0.8 points of full attention on average, single-needle retrieval is 100% for DisSparse and full attention at both lengths, and DisSparse matches Twilight to within 0.4 points (Table 9, Appendix B).

**Sparsity Analyzer/Selection Performance.** Figure 9 isolates the algorithmic contribution: both methods are run in matching two-kernel decompositions (GEMV + top-$p$ post-pass), with Quest-style block selection as the first stage. We compare the end-to-end sparse analyzer latency of Twilight and DisSparse and focus on the sparsity analyzer because, in our two-stream pipeline, this component dominates the selection-stream latency and bounds the achievable overlap with attention computation as batch size and sequence length grow (Figure 3). DisSparse speeds up selection by 1.17× to 1.80× across all configurations. The speedup grows with batch size (more post-pass parallelism) and is

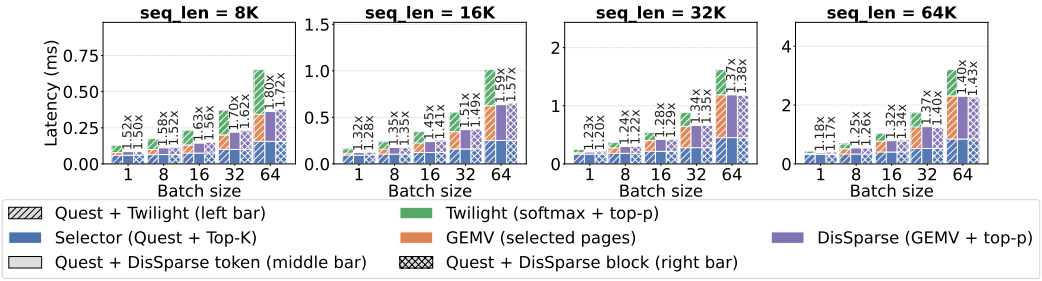



Figure 9: Sparsity analyzer latency for Quest+Twilight vs Quest+DisSparse. DisSparse replaces Twilight's softmax + binary-search post-pass with more efficient GEMV for score calculation and top-$p$ kernel in Algorithm 1.

largest at short sequences: at 8K it reaches $1.80\times$ (token level) and $1.72\times$ (block level) at batch 64, while at 64K it ranges from $1.17\times$ to $1.43\times$ as GEMV takes a larger share of the latency and leaves less post-pass time for DisSparse to optimize. DisSparse's algorithms are implemented in CUDA and OpenAI Triton [69] based on the design details in Section 4.

**Token Relocation.** Table 4 summarizes the impact of token relocation across three models and two long-context benchmarks (more details in Tables 8, 10, and 11 in Appendix B). Relocation reduces the average critical block count by 7.6–14.1% across the eight (model, benchmark, block size) combinations we tested, with the largest reductions on LongChat-7B (14.1%, MHA) and Llama-3.1-8B at block size 32 (13.2%, GQA ratio 4). Accuracy is preserved: on LongBench the largest drop is $-0.80$ points (within run-to-run variance), and on LongBenchV2 relocation maintains or improves accuracy (0 to $+3.03$ points across Llama-3.1-8B and Qwen2.5-7B). We use a band size of $d_{\text{band}} = 4096$ tokens (yielding $R = \lceil sequence_length/d_{\text{band}} \rceil$ bands per sequence) in Algorithm 2, which performs well across our tested workloads.

### 5.2 System Performance

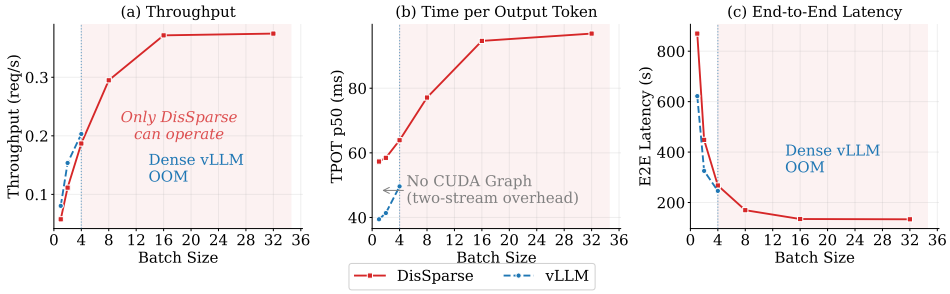
**Setup.** We integrate DisSparse into vLLM with disaggregated inference and evaluate on LongBenchV2 samples up to 32K tokens, using two A100-40GB GPUs for prefill and decode. The decode engine's memory is partitioned among model weights, KV cache (FP16 primary + INT4 replica), and an NCCL buffer that holds the KV cache during asynchronous transfer from the prefill engine. For dense vLLM, the buffer must accommodate FP16 KV caches for the entire batch, whereas DisSparse transfers only the INT4 key replica during the first decode step, $8\times$ smaller, and offloads the FP16 cache to CPU memory in the background. Subsequent decode steps fetch only the blocks selected by the sparse analyzer.

Table 4: Reduction: percentage decrease in average critical block counts ($p$=0.95) with DisSparse. Acc. $\Delta$ shows the change in average benchmark score from No Reloc. to Reloc. (negative: relocation lost accuracy).

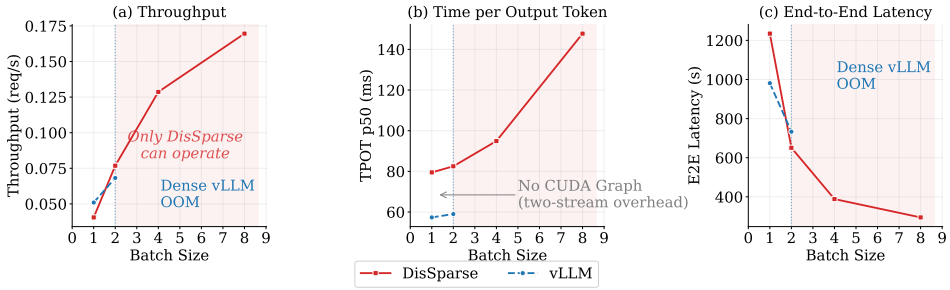
| Model | Benchmark | Block Size | Mean Reduction | Acc. $\Delta$ |
|---|---|---|---|---|
| Llama-3.1-8B | LongBench | 16 | 11.4% | $-0.17$ |
| | LongBench | 32 | 13.2% | $-0.44$ |
| | LongBenchV2 | 16 | 9.4% | $+0.79$ |
| | LongBenchV2 | 32 | 12.4% | 0 |
| Qwen2.5-7B | LongBench | 16 | 7.9% | $-0.17$ |
| | LongBench | 32 | 7.6% | $-0.80$ |
| | LongBenchV2 | 16 | 8.1% | $+3.03$ |
| LongChat-7B | LongBench | 16 | 14.1% | $-0.40$ |

**Results.** Figure 10 compares DisSparse against dense vLLM on Llama-3.1-8B-Instruct (GQA) and Phi-3-mini-128k-instruct (MHA). Dense vLLM exhausts GPU memory at batch sizes 4 and 2, respectively, while DisSparse sustains batch sizes up to 32 and 8 under the same memory budget, achieving $2\times$ and $2.5\times$ higher peak throughput. At small batch sizes, dense vLLM is approximately $1.5\times$ faster in TPOT due to CUDA Graph capture. DisSparse's two-stream pipeline is incompatible with vLLM's single-stream CUDA Graph implementation, so we disable graph capture, exposing the kernel launch overhead. We view this as an engineering limitation rather than a fundamental one[63]. Additionally, the implementation in vLLM does not employ Quest as the top-$k$ page filter; hence, DisSparse has to do a large GEMV on the whole INT4 cache. Notably, at batch size 2 on Phi-3-mini, DisSparse already outperforms dense vLLM on end-to-end latency: the MHA architecture's larger KV cache makes the dense baseline KV-cache-bound at this batch size, while DisSparse's sparse

selection reduces the effective KV pressure. The substantial increase in batch size can be related to the reduction in KV cache network traffic from prefill node to decode node. Unlike dense attention, the sparse attention fetches only critical tokens/pages and DisSparse's top-$p$ selection keeps the accuracy high.




(a) Llama-3.1-8B Performance




(b) Phi-3-Mini Performance

Figure 10: Performance of DisSparse vs. vLLM[5] with Llama 3.1-8B-Instruct[47] and Phi-3-mini-128k-instruct[49] models on LongBenchV2 samples[11] (up to 32k sequence length). Each system is evaluated on 3 metrics: throughput, TPOT, and end-to-end latency vs. batch size.

## 6 Conclusion

The dominance of top-$k$ in dynamic sparse attention is a consequence of system constraints, not algorithmic merit: fixed budgets align with deterministic memory allocation, but they mismatch the actual variability of attention distributions across heads, layers, and decode steps. We present DisSparse, an algorithm-system co-design that integrates top-$p$ selection into vLLM end-to-end by decoupling block selection from the scheduler's critical path, hiding selection latency behind layer computation, and replacing multi-pass selection with a single-pass histogram kernel. Across multiple models and both long- and medium-context benchmarks, DisSparse matches dense-attention accuracy while sustaining up to $8\times$ larger batch sizes than dense vLLM under the same memory budget, evidence that the systems-level barriers to sparse attention can be lifted without algorithmic compromise. The remaining small-batch gap stems from CUDA Graph compatibility, which multi-stream capture should close, and missing Quest as the first layer of sparsity. We discuss the limitations in Appendix C.

## Acknowledgments and Disclosure of Funding

We thank the anonymous reviewers for their feedback. This work was partially supported by the National Science Foundation under Grants 2512416, 2339317 (CAREER), and 2448156, and in part by the Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation, and workforce development (`https://www.cyberinitiative.org`). We also thank Google Cloud Platform and the UCR HPCC for hardware resources.

## References


[1] Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B. and Han, S. (2024) Query-aware sparsity for efficient long-context LLM inference. *arXiv preprint arXiv:2406.10774*.

[2] Chen, R., et al. (2024) ArkVale: Efficient Generative LLM Inference with Recallable Key-Value Eviction. *Advances in Neural Information Processing Systems*, 37, pp. 113134–113155.

[3] Xiao, C., et al. (2024) InfLLM: Training-free Long-Context Extrapolation for LLMs with an Efficient Context Memory. *Advances in Neural Information Processing Systems*, 37, pp. 119638–119661.

[4] Jiang, H., et al. (2024) Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37, pp. 52481–52515.

[5] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., ... & Stoica, I. (2023). Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (pp. 611–626).

[6] Zheng, L., et al. (2024) SGLang: Efficient Execution of Structured Language Model Programs. *Advances in Neural Information Processing Systems (NeurIPS)*, 37, pp. 62557–62583.

[7] Lin, C., et al. (2025) Twilight: Adaptive Attention Sparsity with Hierarchical Top-$p$ Pruning. *arXiv preprint arXiv:2502.02770*.

[8] Zhang, J., Wei, J., Huang, H., Zhang, P., Zhu, J., and Chen, J. (2024) SageAttention: Accurate 8-bit Attention for Plug-and-Play Inference Acceleration. *arXiv preprint arXiv:2410.02367*.

[9] Shi, X., et al. (2025) Nexus: Proactive Intra-GPU Disaggregation of Prefill and Decode in LLM Serving. *arXiv preprint arXiv:2507.06608*.

[10] Zhong, Y., et al. (2024) DistServe: Disaggregating Prefill and Decoding for Goodput-Optimized Large Language Model Serving. *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210.

[11] Bai, Y., et al. (2024) LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks. *arXiv preprint arXiv:2412.15204*.

[12] Bai, Y., et al. (2024) LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 3119–3137.

[13] Artificial Analysis Team (2025) Artificial Analysis Long Context Reasoning Benchmark (LCR). Artificial Analysis, Inc. `https://artificialanalysis.ai/`.

[14] Wu, W., et al. (2024) Retrieval Head Mechanistically Explains Long-Context Factuality. *arXiv preprint arXiv:2404.15574*.

[15] Xiao, G., et al. (2024) DuoAttention: Efficient Long-Context LLM Inference with Retrieval and Streaming Heads. *arXiv preprint arXiv:2410.10819*.

[16] Jeong, J. and Ahn, J. (2025) Accelerating LLM Serving for Multi-Turn Dialogues with Efficient Resource Management. *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Vol. 2, pp. 1–15.

[17] Yang, A., et al. (2025) Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*.

[18] Huang, L., et al. (2025) Towards Agentic AI for Science: Hypothesis Generation, Comprehension, Quantification, and Validation. *Companion Proceedings of the ACM on Web Conference (WWW)*, pp. 1639–1642.

[19] Zhang, H., et al. (2025) SPAD: Specialized Prefill and Decode Hardware for Disaggregated LLM Inference. *arXiv preprint arXiv:2510.08544*.

[20] Li, X., et al. (2025) KVTuner: Sensitivity-Aware Layer-Wise Mixed-Precision KV Cache Quantization for Efficient and Nearly Lossless LLM Inference. *arXiv preprint arXiv:2502.04420*.

[21] Zhang, Y., Hu, Y., Zhao, R., Lui, J. C. S. and Chen, H. (2025) DiffKV: Differentiated Memory Management for Large Language Models with Parallel KV Compaction. *Proceedings of the 30th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 431–445.

[22] Zandieh, A., Daliri, M., Hadian, M. and Mirrokni, V. (2026) TurboQuant: Online Vector Quantization with Near-Optimal Distortion Rate. *International Conference on Learning Representations (ICLR)*.

[23] Ge, S., et al. (2024) Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs. *International Conference on Learning Representations (ICLR)*.

[24] Zaheer, M., et al. (2020) Big Bird: Transformers for Longer Sequences. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, pp. 17283–17297.

[25] Beltagy, I., Peters, M. E. and Cohan, A. (2020) Longformer: The Long-Document Transformer. *arXiv preprint arXiv:2004.05150*.

[26] Chen, Z., et al. (2025) MagicPIG: LSH Sampling for Efficient LLM Generation. *International Conference on Learning Representations (ICLR)*.

[27] Ribar, L., et al. (2024) SparQ Attention: Bandwidth-Efficient LLM Inference. *International Conference on Machine Learning (ICML)*.

[28] Park, J., Kim, S., Cheung, A. and Stoica, I. (2026) Qrita: High-performance Top-k and Top-p Algorithm for GPUs using Pivot-based Truncation and Selection. *arXiv preprint arXiv:2602.01518*.

[29] Holtzman, A., Buys, J., Du, L., Forbes, M. and Choi, Y. (2020) The Curious Case of Neural Text Degeneration. *International Conference on Learning Representations (ICLR)*.

[30] Vaswani, A., et al. (2017) Attention is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30.

[31] Ainslie, J., Lee-Thorp, J., De Jong, M., Zemlyanskiy, Y., Lebrón, F. and Sanghai, S. (2023) GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4895–4901.

[32] Shazeer, N. (2019) Fast Transformer Decoding: One Write-Head is All You Need. *arXiv preprint arXiv:1911.02150*.

[33] Yu, G.-I., et al. (2022) Orca: A Distributed Serving System for Transformer-Based Generative Models. *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538.

[34] Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. (2023) H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, pp. 34661–34710.

[35] Xiao, G., Tian, Y., Chen, B., Han, S. and Lewis, M. (2024) Efficient Streaming Language Models with Attention Sinks. *International Conference on Learning Representations (ICLR)*.

[36] Liu, Z., et al. (2023) Déjà Vu: Contextual Sparsity for Efficient LLMs at Inference Time. *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pp. 22137–22176.

[37] Liu, A., et al. (2025) DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. *arXiv preprint arXiv:2512.02556*.

[38] Dettmers, T., Lewis, M., Belkada, Y. and Zettlemoyer, L. (2022) LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *Advances in Neural Information Processing Systems (NeurIPS)*, 35, pp. 30318–30332.

[39] Guo, C., et al. (2023) Olive: Accelerating Large Language Models via Hardware-Friendly Outlier-Victim Pair Quantization. *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–15.

[40] Nagel, M., et al. (2021) A White Paper on Neural Network Quantization. *arXiv preprint arXiv:2106.08295*.

[41] Lin, Y., et al. (2025) QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving. *Proceedings of Machine Learning and Systems (MLSys)*, 7.

[42] Zhao, Y., et al. (2024) Atom: Low-bit Quantization for Efficient and Accurate LLM Serving. *Proceedings of Machine Learning and Systems (MLSys)*, 6, pp. 196–209.

[43] Liu, Z., et al. (2023) Scissorhands: Exploiting the Persistence of Importance Hypothesis for LLM KV Cache Compression at Test Time. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, pp. 52342–52364.

[44] Li, Y., et al. (2024) SnapKV: LLM Knows What You are Looking for Before Generation. *Advances in Neural Information Processing Systems (NeurIPS)*, 37, pp. 22947–22970.

[45] NVIDIA. (2026) NVIDIA Collective Communications Library (NCCL). Available at: `https://developer.nvidia.com/nccl` (Accessed: 26 April 2026).

[46] Zhou, Q., Yin, P., Zuo, P. and Cheng, J. (2025) Progressive Sparse Attention: Algorithm and System Co-design for Efficient Attention in LLM Serving. *arXiv preprint arXiv:2503.00392*.

[47] Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Grattafiori, A., et al. (2024) The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*.

[48] Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., et al. (2024) Qwen2 Technical Report. *arXiv preprint arXiv:2407.10671*.

[49] Abdin, M., et al. (2024) Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. *arXiv preprint arXiv:2404.14219*.

[50] Li, D., et al. (2023) How Long Can Open-Source LLMs Truly Promise on Context Length? *LMSYS Org Blog*. Available at: https://lmsys.org/blog/2023-06-29-longchat/.

[51] Yang, S., Sheng, Y., Gonzalez, J. E., Stoica, I. and Zheng, L. (2024) Post-Training Sparse Attention with Double Sparsity. *arXiv preprint arXiv:2408.07092*.

[52] Men, X., et al. (2025) Shortgpt: Layers in large language models are more redundant than you expect. *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204.

[53] Lee, W., et al. (2024) InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172.

[54] Dao, T. (2024) FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *International Conference on Learning Representations (ICLR)*.

[55] Levy, N. (2026) Dynamic Sparse Attention: Access Patterns and Architecture. *arXiv preprint arXiv:2603.13430*.

[56] Sun, H., et al. (2024) Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*.

[57] Desai, A., et al. (2025) vAttention: Verified Sparse Attention. *arXiv preprint arXiv:2510.05688*.

[58] Wu, W., et al. (2025) Tokenselect: Efficient long-context inference and length extrapolation for llms via dynamic token-level kv cache selection. *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21275–21292.

[59] Cobbe, K., et al. (2021) Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

[60] Reddy, S., Chen, D., and Manning, C. D. (2019) Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 249–266.

[61] Rae, J. W., et al. (2019) Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*.

[62] Ni, W., et al. (2026) Double-P: Hierarchical Top-P Sparse Attention for Long-Context LLMs. *arXiv preprint arXiv:2602.05191*.

[63] lambda7xx. (2025) [Bug]: How to make vLLM support multi stream torch compile and each stream capture cuda graph. Issue #30211. *vLLM GitHub Repository*. `https://github.com/vllm-project/vllm/issues/30211`

[64] Yang, A., et al. (2025) Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*.

[65] Touvron, H., et al. (2023) Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[66] vLLM Team. (2026) Disaggregated Prefill. *vLLM Documentation*. `https://docs.vllm.ai/en/latest/features/disagg_prefill/`

[67] bsantraigi. (2025) Batched top-k and top-p/nucleus sampling in PyTorch! *GitHub Gist*. `https://gist.github.com/bsantraigi/5752667525d88d375207f099bd78818b`

[68] Sheng, Y., et al. (2023) Flexgen: High-throughput generative inference of large language models with a single GPU. *International Conference on Machine Learning*, pp. 31094–31116.

[69] Tillet, P., et al. (2019) Triton: an intermediate language and compiler for tiled neural network computations. *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19.

[70] Su, J., et al. (2024) Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, vol. 568, p. 127063.

[71] Hsieh, C.-P., et al. (2024) RULER: What's the Real Context Size of Your Long-Context Language Models? *Conference on Language Modeling (COLM)*.

[72] Feng, Y., Lv, J., Cao, Y., Xie, X., and Zhou, S. K. (2025) Ada-KV: Optimizing KV Cache Eviction by Adaptive Budget Allocation for Efficient LLM Inference. *Advances in Neural Information Processing Systems (NeurIPS)*.

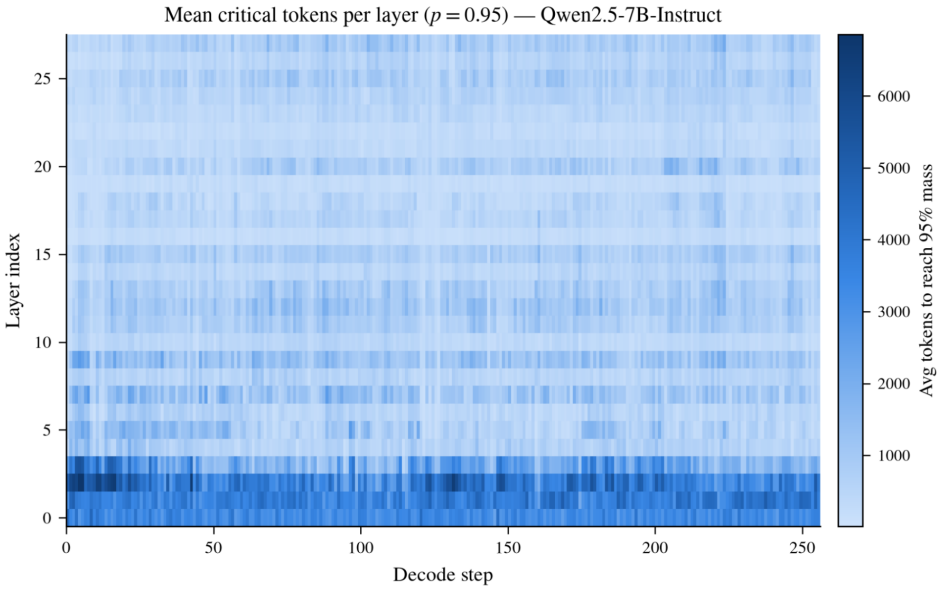
[73] Singhania, P., Singh, S., He, S., Feizi, S., and Bhatele, A. (2024) Loki: Low-rank Keys for Efficient Sparse Attention. *Advances in Neural Information Processing Systems (NeurIPS)*.

[74] Mu, J., Huang, H., Zhang, J., Yu, M., Wang, T., and Li, Y. (2025) SALS: Sparse Attention in Latent Space for KV Cache Compression. *Advances in Neural Information Processing Systems (NeurIPS)*.

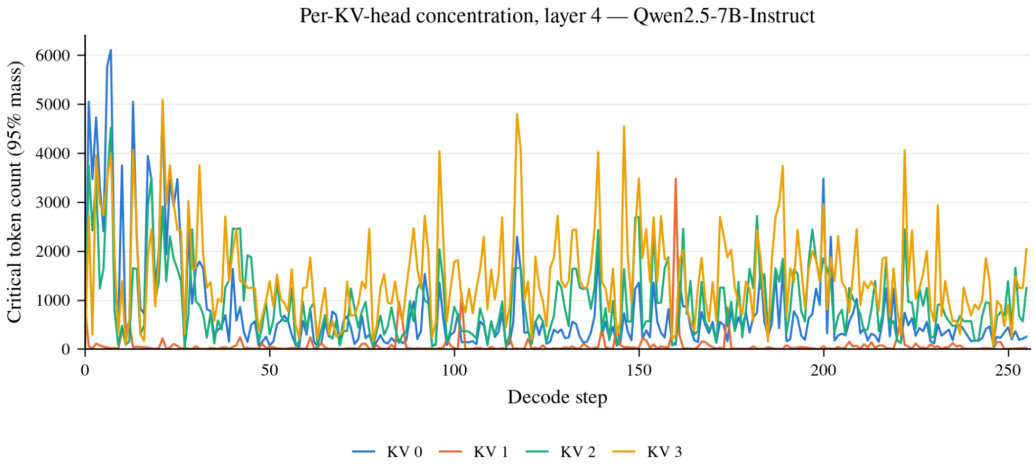
# A Critical Token Dynamics on Additional Models

In this section of the appendix, Figures 11, 12, and 13 are presented to extend the analysis of Figure 5 and 1 and demonstrate the generality of the observation for DisSparse's motivation, which is the dynamic variation in critical KV cache tokens/blocks. We extend the analysis to Qwen2.5-7B-Instruct [17], Qwen3-32B, and Gemma-2-27B-it with two kinds of plots: per-layer average critical token heatmap and per-KV-head variation in critical token count over 256 decode steps on WikiText samples. Across layers, earliest layers need more tokens compared to later layers (layers 0–3 in Qwen2.5-7B, 0–6 in Qwen3-32B, and 0–4 in Gemma-2-27B) and later layers have fewer but varying average counts of critical tokens. Across KV heads, the critical token count varies significantly. For example, in layer 8 of Qwen3-32B, KV head 7 reaches about 3500 tokens, while KV heads 0, 1, and 6 need a comparatively small set of tokens.

Gemma-2-27B alternates local (sliding-window) and global attention layers, and the two layer types behave differently (Figure 13, bottom). In local layer 6, every KV head stays below about 1000 tokens, well under the 4096-token window. In global layer 5, one KV head needs roughly 1000–3400 tokens at most steps, while the others stay near zero. Together, these results show that the mismatch between a fixed $k$ and attention dynamics described in Section 3 extends to these models, including one with a hybrid local and global attention design.

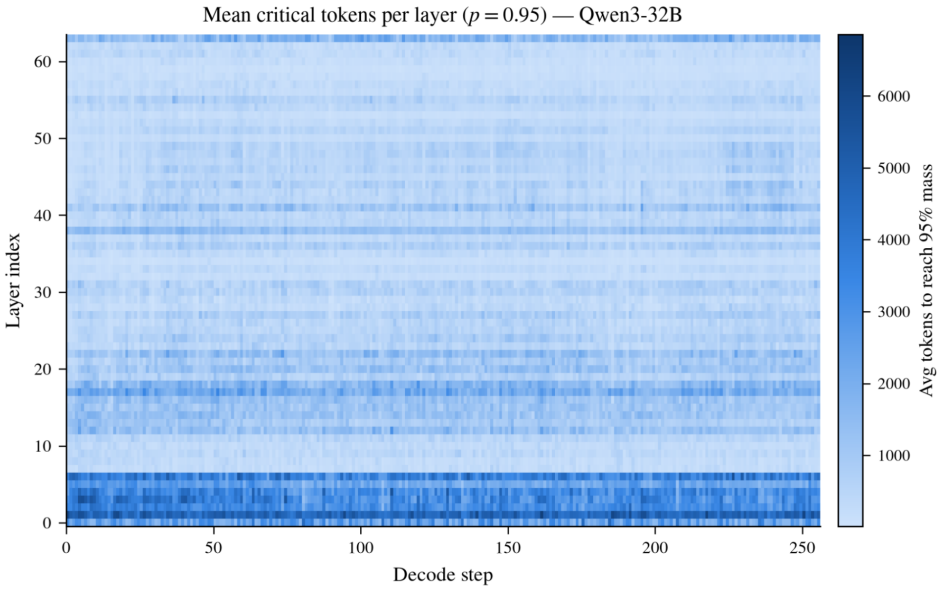



(a) Mean critical tokens per layer.

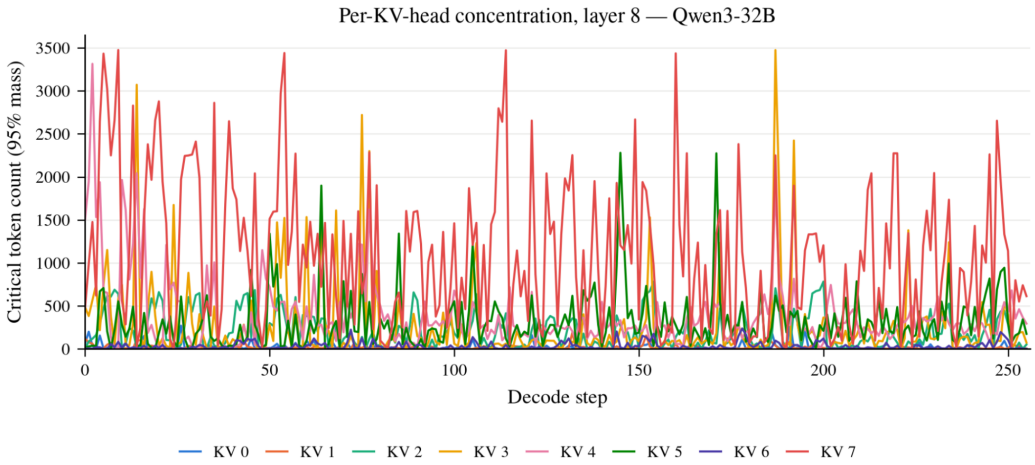



(b) Critical tokens per KV head, layer 4.

Figure 11: Critical token counts ($p = 0.95$) across decode steps for Qwen2.5-7B-Instruct.

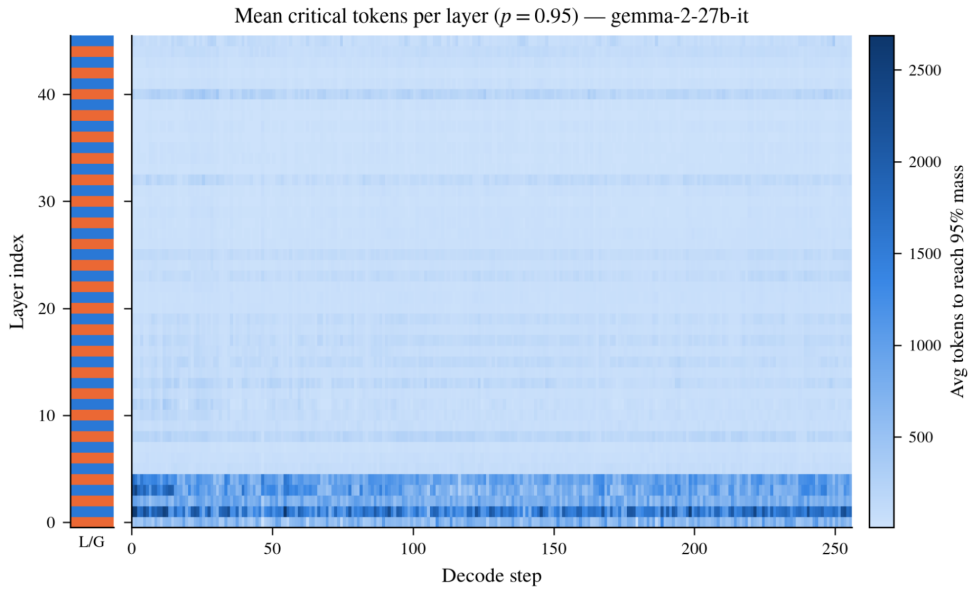



(a) Mean critical tokens per layer.

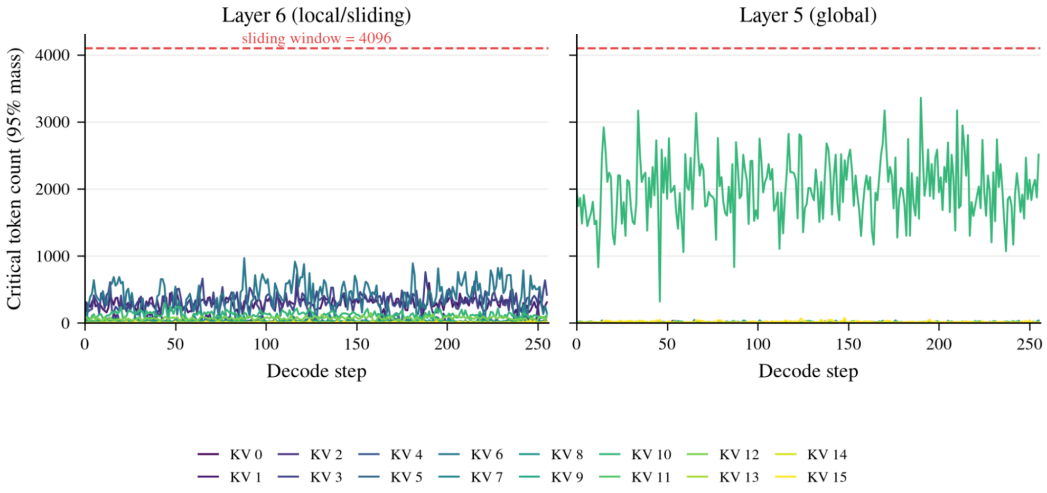



(b) Critical tokens per KV head, layer 8.

Figure 12: Critical token counts ($p = 0.95$) across decode steps for Qwen3-32B.




(a) Mean critical tokens per layer. The L/G strip marks local (orange) and global (blue) layers.




(b) Critical tokens per KV head in a local and a global layer.

Figure 13: Critical token counts ($p = 0.95$) across decode steps for Gemma-2-27B-it.

# B Detailed Results

Table 5: Per-dataset accuracy on LongBench (A100). **Bold** indicates the best result within each model section (per dataset). Green and red percentages denote change relative to Full Attention. Top-$p$ configurations use $p = 0.95$ for *LlaMA-3.1-8B-Instruct* and $p = 0.85$ for *LongChat-7B-v1.5-32k*.

| | | Single-Doc. QA | | Multi-Doc. QA | | | Summarization | | | Few | Syn. | Code | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | **Budget** | Qasper | MF-en | HotpotQA | 2WikiMQA | Musique | GovRep. | QMSum | M-News | TriviaQA | PR-en | LCC | Repo. | **Avg.** |
| *LongChat-7B-v1.5-32k* | | | | | | | | | | | | | | |
| Full Attn. | 32k | 29.07 | 41.45 | 31.33 | 21.37 | 12.94 | 30.88 | 23.01 | 25.99 | 84.02 | 29.13 | 49.41 | 54.63 | 36.10 |
| Quest | 256 | 25.64 | 32.31 | 23.50 | 19.93 | 7.36 | 22.53 | 21.20 | 24.95 | 68.03 | 32.10 | 45.66 | 44.27 | 30.62 |
| | 1024 | 31.19 | 41.69 | 30.83 | 21.98 | 9.98 | 29.79 | 22.94 | 26.29 | 84.99 | 32.96 | 48.31 | 52.99 | 36.16 |
| | 4096 | 29.36 | 42.04 | 33.33 | 21.55 | 13.07 | 31.38 | 23.10 | 26.35 | 85.15 | 30.09 | 49.86 | 54.53 | 36.65 |
| | 8192 | 28.94 | 41.04 | 33.66 | 21.12 | 13.34 | 31.03 | 23.26 | 26.38 | 85.49 | 28.66 | 49.71 | 54.58 | 36.43 |
| | + Twi. (137) | 31.51 | 42.60 | 31.99 | 22.39 | 13.31 | 32.05 | 23.03 | 26.23 | 85.72 | 32.00 | 51.44 | 55.69 | 37.33 +3.4% |
| DS | 256 | 27.89 | 39.15 | 27.42 | 18.68 | 9.22 | 29.53 | 22.02 | 25.59 | 84.75 | 30.57 | 48.77 | 52.49 | 34.67 |
| | 1024 | 30.13 | 40.62 | 31.21 | 19.69 | 7.18 | 26.69 | 23.19 | 26.41 | 83.99 | 30.09 | 49.91 | 54.51 | 35.30 |
| | 4096 | 28.55 | 41.24 | 32.90 | 21.29 | 7.97 | 29.53 | 22.99 | 26.45 | 84.12 | 28.66 | 49.48 | 54.49 | 35.64 |
| | 8192 | 29.05 | 41.42 | 32.16 | 20.66 | 12.34 | 30.29 | 22.74 | 26.33 | 84.41 | 29.13 | 49.57 | 54.34 | 36.04 |
| | + Twi. (126) | 31.89 | 43.20 | **35.08** | 22.89 | 13.67 | 31.72 | 23.25 | 26.22 | **86.08** | **33.91** | **51.60** | 56.25 | **37.98** +5.2% |
| Twilight | Avg. 146 | 31.30 | 43.22 | 33.98 | 23.09 | 13.75 | 32.03 | **23.39** | 26.20 | 85.93 | 32.96 | 51.55 | 56.10 | 37.79 +4.7% |
| Quest+DisSparse (token) | Avg. 141 | **32.91** | 42.80 | 33.22 | **24.80** | 13.78 | 31.73 | 22.68 | **26.79** | 84.62 | 31.00 | 50.93 | 56.69 | 37.66 +4.3% |
| DisSparse(token) | Avg. 170 | 31.37 | **43.79** | 34.79 | 23.51 | **15.00** | **32.28** | 22.36 | 26.15 | 85.50 | 31.00 | 50.72 | **56.98** | 37.79 +4.7% |
| *LLaMA-3.1-8B-Instruct* | | | | | | | | | | | | | | |
| Full Attn. | 128k | 46.17 | 53.33 | 55.36 | 43.95 | 27.08 | 35.01 | 25.24 | 27.37 | 91.18 | 99.50 | 62.17 | 57.76 | 52.01 |
| Quest | 256 | 25.20 | 38.10 | 29.10 | 29.10 | 9.30 | 26.50 | 19.70 | 26.50 | 66.20 | 95.50 | 53.00 | 47.10 | 38.80 |
| | 1024 | 39.90 | 50.90 | 47.50 | 37.40 | 22.90 | 33.70 | 23.80 | **27.50** | 80.90 | 99.50 | 59.90 | 56.70 | 48.40 |
| | 4096 | 45.80 | 51.10 | 51.60 | 37.90 | 24.80 | 34.70 | 25.10 | **27.50** | 89.80 | 99.50 | 63.00 | 57.30 | 50.70 |
| | 8192 | 45.10 | 53.00 | 54.90 | 44.80 | 25.40 | 34.60 | **25.60** | 27.30 | 91.40 | 99.50 | 62.90 | 57.50 | 51.80 |
| | + Twi. (489) | 45.94 | 53.15 | 53.93 | 46.04 | 25.10 | 34.20 | 25.35 | 27.20 | 90.65 | 99.50 | 63.00 | 60.20 | 52.00 -0.0% |
| DS | 256 | 38.24 | 49.58 | 43.38 | 31.98 | 15.52 | 33.40 | 24.06 | 26.86 | 84.41 | 99.50 | 53.28 | 48.64 | 45.74 |
| | 1024 | 42.97 | **54.65** | 51.75 | 33.92 | 20.39 | 34.50 | 24.92 | 26.71 | **92.81** | 99.50 | 62.66 | 48.37 | 49.43 |
| | 4096 | 43.50 | 53.17 | 54.21 | 44.70 | 23.14 | **34.73** | 25.40 | 26.71 | 92.78 | 99.50 | 62.59 | 51.31 | 50.98 |
| | 8192 | 43.82 | 53.71 | 54.19 | 45.13 | 23.72 | 34.27 | 24.98 | 26.69 | 91.61 | **100.00** | 62.40 | 52.87 | 51.14 |
| | + Twi. (446) | 43.08 | 52.89 | 54.68 | 44.86 | 24.88 | 34.09 | 25.20 | 27.00 | 91.20 | **100.00** | **63.95** | 58.93 | 51.73 -0.5% |
| Twilight | Avg. 510 | 46.00 | 53.10 | 54.00 | 46.00 | 25.10 | 34.20 | 25.40 | 27.20 | 90.60 | 99.50 | 63.00 | 60.20 | 52.00 -0.0% |
| Quest+DisSparse(token) | Avg. 520 | **46.60** | 52.50 | 54.80 | **46.50** | 24.80 | 34.50 | 25.40 | 27.40 | 90.30 | 99.50 | 63.30 | 59.70 | 52.10 +0.2% |
| DisSparse(token) | Avg. 610 | 45.60 | 52.90 | **55.10** | 44.70 | **28.00** | 33.90 | 25.20 | 27.20 | 90.60 | 99.50 | 63.00 | **60.40** | **52.20** +0.4% |

Table 6: Per-dataset accuracy on LongBench (RTX 6000 Ada). **Bold** indicates the best result within each model section (per dataset). All top-$p$ configurations use $p = 0.95$. Green and red percentages denote change relative to Full Attention.

| | | Single-Doc. QA | | Multi-Doc. QA | | | Summarization | | | Few | Syn. | Code | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | **Budget** | Qasper | MF-en | HotpotQA | 2WikiMQA | Musique | GovRep. | QMSum | M-News | TriviaQA | PR-en | LCC | Repo. | **Avg.** |
| *LongChat-7B-v1.5-32k* | | | | | | | | | | | | | | |
| Full Attn. | 32k | 27.23 | 41.45 | 30.45 | 23.53 | **13.02** | 31.42 | 22.69 | 26.52 | 83.96 | 31.50 | 54.65 | 58.04 | 37.04 |
| Quest | 256 | 23.31 | 33.44 | 20.15 | 21.68 | 7.39 | 22.88 | 21.39 | 25.79 | 69.62 | **34.50** | 50.17 | 48.56 | 31.57 -14.8% |
| | 1024 | 28.91 | 40.89 | 30.62 | 22.27 | 11.41 | 31.18 | 22.22 | 26.46 | **84.72** | 34.50 | 52.21 | 57.42 | 36.90 -0.4% |
| | 4096 | 28.68 | 42.18 | 30.31 | 23.08 | 11.19 | 31.56 | 22.26 | **26.69** | 83.97 | 29.00 | 54.70 | 58.47 | 36.84 -0.5% |
| | 8192 | 27.47 | 41.36 | 31.28 | 23.65 | 11.71 | 31.33 | 22.36 | 26.38 | 83.91 | 28.00 | 54.41 | 57.93 | 36.65 -1.1% |
| | + Twi. (398) | **29.48** | **42.85** | 31.16 | **24.11** | 12.36 | 31.74 | 22.56 | 26.45 | 84.49 | 30.50 | 54.74 | **58.73** | 37.43 +1.1% |
| | + DisSparse (token) (424) | 29.11 | 42.60 | 31.49 | 23.84 | 12.73 | 31.77 | 22.61 | 26.30 | 84.02 | 32.00 | 54.61 | 58.38 | **37.45** +1.1% |
| | + DisSparse (block) (982) | 28.94 | 40.34 | **32.57** | 23.93 | 12.35 | 31.72 | **22.76** | 26.54 | 83.97 | 33.00 | 53.89 | 58.52 | 37.38 +0.9% |
| Twilight | Avg. 468 | 28.66 | 41.44 | 31.26 | 23.41 | 12.78 | **32.01** | 22.36 | 26.45 | 83.54 | 33.00 | 55.25 | 58.60 | 37.40 +1.0% |
| DisSparse(token) | Avg. 500 | 28.46 | 40.40 | 31.31 | 23.47 | 12.58 | 31.71 | 22.59 | 26.47 | 83.81 | 34.50 | **55.29** | 58.61 | 37.43 +1.1% |
| *LLaMA-3.1-8B-Instruct* | | | | | | | | | | | | | | |
| Full Attn. | 128k | 46.10 | **52.98** | **56.17** | 43.65 | **28.41** | **34.85** | 25.19 | **27.42** | 91.47 | **99.50** | 64.51 | 59.60 | 52.49 |
| Quest | 256 | 24.71 | 36.54 | 30.73 | 28.81 | 13.51 | 26.56 | 19.58 | 26.45 | 64.96 | 92.00 | 54.43 | 48.38 | 38.89 -25.9% |
| | 1024 | 39.59 | 49.91 | 47.65 | 38.96 | 20.66 | 33.42 | 23.85 | 27.29 | 81.53 | 99.50 | 60.66 | 57.52 | 48.38 -7.8% |
| | 4096 | **47.15** | 51.81 | 52.16 | 37.67 | 23.72 | 34.47 | 25.23 | 27.41 | 89.82 | 99.50 | 64.85 | 59.85 | 51.14 -2.6% |
| | 8192 | 46.39 | 52.39 | 54.83 | 45.07 | 25.82 | 34.75 | 25.47 | 27.41 | **91.73** | 99.50 | 64.69 | 59.49 | 52.29 -0.4% |
| | + Twi. (489) | 46.70 | 52.37 | 53.93 | 44.65 | 24.49 | 34.43 | **25.48** | 27.35 | 90.14 | 99.50 | 65.05 | 62.23 | 52.19 -0.6% |
| | + DisSparse (token) (519) | 46.59 | 52.25 | 54.04 | **46.03** | 25.62 | 34.75 | 25.31 | 27.08 | 90.60 | 99.50 | 64.95 | 61.76 | 52.37 -0.2% |
| | + DisSparse (block) (1138) | 46.10 | 52.65 | 55.39 | 43.83 | 27.86 | 33.73 | 25.01 | 27.24 | 91.47 | 99.50 | 65.00 | 62.22 | 52.50 +0.0% |
| Twilight | Avg. 603 | 46.01 | 52.70 | 54.87 | 44.22 | 28.33 | 34.32 | 25.13 | 26.91 | 91.14 | 99.50 | 64.82 | **62.64** | 52.55 +0.1% |
| DisSparse(token) | Avg. 636 | 45.61 | 52.78 | 54.01 | 45.59 | 28.40 | 34.09 | 25.11 | 27.12 | 91.05 | 99.50 | **65.31** | 62.25 | **52.57** +0.2% |

The 11 subtasks are the eight needle-in-a-haystack variants (single, multi-key, multi-value, and multi-query) plus variable tracking and common- and frequent-word extraction. On the eight needle subtasks, the DisSparse average is within 0.5 points of full attention at both lengths. Common-word extraction drops sharply at 32K for every configuration, including full attention (1.2%): the model emits a generic stopword list regardless of the gold words, so this reflects a model limitation and not sparsity.

Table 7: Per-dataset accuracy of Qwen3-32B on LongBench (RTX 6000 Ada). **Bold** indicates the best result per dataset. Green and red percentages denote change relative to Full Attention. The thinking mode is disabled for the experiment with *enable_thinking=False*. All top-$p$ configurations use $p = 0.95$.

| | | Single-Doc. QA | | Multi-Doc. QA | | | Summarization | | | Few | Syn. | Code | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | **Budget** | Qasper | MF-en | HotpotQA | 2WikiMQA | Musique | GovRep. | QMSum | M-News | TriviaQA | PR-en | LCC | Repo. | **Avg.** |
| | | | | | | *Qwen3-32B* | | | | | | | | |
| Full Attn. | 32k | 44.03 | 48.42 | 56.85 | 48.51 | **37.67** | 32.85 | 23.76 | 25.35 | **88.82** | **99.50** | **68.94** | 66.10 | 53.40 |
| Quest | 256 | 27.56 | 34.51 | 25.50 | 24.34 | 11.50 | 24.70 | 22.18 | 24.13 | 65.84 | 77.00 | 53.80 | 43.25 | 36.19 -32.2% |
| | 1024 | 36.90 | 41.94 | 43.73 | 35.74 | 23.62 | 32.18 | 23.68 | **25.59** | 78.42 | 85.00 | 61.87 | 60.25 | 45.74 -14.3% |
| | 4096 | 42.24 | 47.05 | 54.81 | 45.90 | 33.86 | 32.56 | **24.20** | 25.19 | 87.86 | 87.00 | 68.58 | 65.72 | 51.25 -4.0% |
| | 8192 | 43.37 | 47.31 | 56.49 | 47.20 | 37.48 | 32.46 | 23.77 | 25.36 | 87.91 | 87.50 | 68.75 | 66.60 | 52.02 -2.6% |
| | + Twi. (429) | 43.50 | 47.84 | 57.38 | 48.94 | 35.09 | 32.70 | 24.07 | 25.28 | 87.91 | 90.00 | 68.65 | **67.00** | 52.36 -1.9% |
| | + DisSparse (token) (346) | 43.01 | 47.77 | 57.42 | 49.37 | 35.92 | 32.54 | 23.64 | 25.26 | 87.74 | 89.00 | 68.62 | 66.20 | 52.21 -2.2% |
| | + DisSparse (block) (1038) | 43.12 | 48.59 | 57.14 | 48.81 | 36.38 | **33.09** | 23.77 | 25.16 | 88.74 | 99.00 | 68.77 | 66.52 | 53.26 -0.3% |
| Twilight | Avg. 535 | **44.67** | **48.78** | 57.04 | 49.13 | 36.39 | 32.86 | 23.73 | 25.31 | 88.74 | 99.00 | 68.59 | 65.84 | 53.34 -0.1% |
| DisSparse(token) | Avg. 449 | 44.52 | 48.47 | **58.05** | **49.87** | 36.03 | 33.08 | 23.37 | 25.30 | 88.74 | 99.00 | 68.74 | 65.74 | **53.41** +0.0% |

Table 8: Comparison of KV Block Usage and Accuracy with and without Relocation for LongBench and LongBenchV2 (Llama-3.1-8B 128K). Results shown for both block sizes, 16 and 32. 'No Reloc.': the KV tokens keep the chronological order in the same way they were generated during prefill. 'Reloc': the KV tokens are relocated after the prefill phase. 'Reduction': Percentage decrease in the critical block count from 'No Reloc' to 'Reloc'.

| | **Block Size: 16** | | | | | **Block Size: 32** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Block Count** | | **Reduction** | **Accuracy** | | **Block Count** | | **Reduction** | **Accuracy** | |
| **Category / Dataset** | **No Reloc.** | **Reloc.** | | **No Reloc.** | **Reloc.** | **No Reloc.** | **Reloc.** | | **No Reloc.** | **Reloc.** |
| *LongBench (by Dataset)* | | | | | | | | | | |
| narrativeqa | 251.2 | 207.7 | 17.3% | 28.05 | 27.93 | 146.4 | 116.6 | 20.4% | 29.73 | 29.21 |
| qasper | 57.9 | 55.3 | 4.5% | 46.55 | 46.22 | 33.0 | 31.3 | 5.0% | 46.84 | 45.82 |
| multifieldqa_en | 68.0 | 63.1 | 7.2% | 52.53 | 53.10 | 39.2 | 36.0 | 8.4% | 52.01 | 51.71 |
| hotpotqa | 116.4 | 102.4 | 12.0% | 54.85 | 55.41 | 67.5 | 58.6 | 13.1% | 51.74 | 52.07 |
| 2wikimqa | 68.3 | 62.9 | 7.9% | 46.52 | 45.06 | 39.2 | 35.8 | 8.8% | 45.73 | 45.29 |
| musique | 141.1 | 123.1 | 12.7% | 24.82 | 26.45 | 81.9 | 70.9 | 13.4% | 22.81 | 21.93 |
| gov_report | 92.5 | 84.0 | 9.2% | 34.46 | 34.41 | 53.9 | 47.4 | 12.0% | 34.00 | 33.56 |
| qmsum | 131.5 | 112.6 | 14.4% | 25.35 | 25.27 | 77.3 | 63.7 | 17.7% | 24.95 | 24.48 |
| multi_news | 32.8 | 32.3 | 1.7% | 27.43 | 27.38 | 19.1 | 18.7 | 1.9% | 27.45 | 27.52 |
| triviaqa | 75.6 | 68.7 | 9.2% | 90.34 | 88.37 | 45.2 | 40.9 | 9.5% | 84.44 | 83.77 |
| passage_retrieval_en | 73.1 | 66.6 | 8.9% | 99.50 | 99.50 | 44.7 | 40.0 | 10.4% | 99.50 | 99.00 |
| lcc | 38.0 | 36.9 | 2.9% | 63.32 | 63.35 | 22.1 | 21.4 | 3.3% | 62.98 | 63.09 |
| repobench-p | 100.4 | 89.5 | 10.8% | 59.67 | 58.67 | 58.8 | 51.1 | 13.0% | 59.01 | 58.03 |
| **Average** | **95.9** | **85.0** | **11.4%** | **50.26** | **50.09** | **56.0** | **48.6** | **13.2%** | **49.32** | **48.88** |
| *LongBenchV2 (by Seq. Len.)* | | | | | | | | | | |
| 8K – 16K | 84.0 | 76.8 | 8.6% | 42.86 | 42.86 | 54.2 | 47.8 | 11.8% | 39.29 | 42.86 |
| 16K – 32K | 124.3 | 114.0 | 8.3% | 35.29 | 37.65 | 80.3 | 70.1 | 12.7% | 29.41 | 28.24 |
| $>$ 32K | 168.7 | 150.8 | 10.6% | 14.29 | 14.29 | 110.8 | 98.9 | 10.8% | 14.29 | 14.29 |
| **Average** | **125.7** | **113.9** | **9.4%** | **30.81** | **31.60** | **76.0** | **66.6** | **12.4%** | **30.83** | **30.83** |

### B.1 Additional Experiments

**Comparison with head-wise adaptive top-$k$.** Table 12 compares DisSparse with Ada-SnapKV, the strongest Ada-KV [72] baseline, on LLaMA-3.1-8B-Instruct. At a similar token budget, DisSparse achieves higher average accuracy. We leave comparison with Expected Attention and KVzip to future work.

**Colocated serving.** We also adapted DisSparse to a colocated server and measured the maximum concurrent requests on LongBenchV2 with LLaMA-3.1-8B on one A100-40GB GPU, with 252 generated tokens on average (Table 13). The gain is smaller than in the disaggregated setting because the dense prefill phase affects colocated performance.

**CUDA Graph.** vLLM captures a single static graph on one stream, which does not support the two-stream pipeline of DisSparse, so we disable graph capture in the main results. Multi-stream capture is supported by CUDA, and is a requested vLLM feature [63]. Disabling `torch.compile` and enabling two-stream CUDA Graph capture improves small-batch TPOT by up to 30% on Llama-3.1-8B and Phi-3-mini. `torch.compile` must be disabled because TorchDynamo cannot treat a `torch.cuda.Stream` as a symbolic input.

Table 9: Average accuracy on 11 RULER subtasks with Llama-3.1-8B-Instruct (50 samples per subtask). All configurations share the same prompts, tokenization, and scoring. DisSparse uses token-level top-$p$

| Configuration | 16K | 32K |
|---|---|---|
| Full Attn. | 94.86 | 89.09 |
| Quest+Twilight | 94.88 | 88.66 |
| Twilight | 94.71 | 88.98 |
| Quest+DisSparse | 95.08 | 88.32 |
| DisSparse | 94.71 | 89.14 |

Table 10: Comparison of KV Block Usage and Accuracy with and without Relocation for LongBench and LongBenchV2 (Qwen2.5-7B-Instruct, 32K). Results shown for both block sizes, 16 and 32. 'No Reloc.': the KV tokens keep the chronological order in the same way they were generated during prefill. 'Reloc': the KV tokens are relocated after the prefill phase. 'Reduction': Percentage decrease in the critical block count from 'No Reloc' to 'Reloc'.

| | **Block Size: 16** | | | | | **Block Size: 32** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Block Count** | | **Reduction** | **Accuracy** | | **Block Count** | | **Reduction** | **Accuracy** | |
| **Category / Dataset** | **No Reloc.** | **Reloc.** | | **No Reloc.** | **Reloc.** | **No Reloc.** | **Reloc.** | | **No Reloc.** | **Reloc.** |
| *LongBench (by Dataset)* | | | | | | | | | | |
| narrativeqa | 225.4 | 201.2 | 10.8% | 19.83 | 16.52 | 125.9 | 113.3 | 10.0% | 16.04 | 13.92 |
| qasper | 60.5 | 58.1 | 4.0% | 38.88 | 39.46 | 34.4 | 32.9 | 4.4% | 39.37 | 39.20 |
| multifieldqa_en | 71.4 | 67.2 | 5.9% | 48.38 | 49.62 | 40.6 | 37.7 | 7.1% | 48.79 | 48.98 |
| hotpotqa | 118.0 | 107.7 | 8.7% | 51.89 | 51.02 | 66.8 | 60.9 | 8.9% | 49.99 | 48.54 |
| 2wikimqa | 71.7 | 67.3 | 6.2% | 34.03 | 34.55 | 41.0 | 38.1 | 7.0% | 35.05 | 34.52 |
| musique | 140.4 | 128.8 | 8.3% | 22.66 | 21.75 | 79.2 | 72.2 | 8.8% | 21.74 | 18.14 |
| gov_report | 100.8 | 93.2 | 7.6% | 29.16 | 29.37 | 56.2 | 52.0 | 7.5% | 28.50 | 27.90 |
| qmsum | 121.2 | 107.7 | 11.2% | 22.27 | 21.53 | 67.9 | 60.8 | 10.5% | 21.95 | 22.28 |
| multi_news | 36.5 | 36.3 | 0.5% | 24.05 | 23.96 | 21.0 | 20.8 | 0.8% | 24.00 | 24.06 |
| triviaqa | 86.3 | 79.9 | 7.4% | 82.06 | 81.21 | 49.6 | 46.8 | 5.8% | 80.42 | 78.71 |
| passage_retrieval_en | 67.5 | 63.8 | 5.5% | 91.88 | 91.92 | 39.1 | 37.2 | 4.9% | 92.03 | 91.54 |
| lcc | 38.0 | 37.1 | 2.2% | 47.34 | 48.39 | 21.9 | 21.4 | 2.2% | 47.37 | 47.79 |
| repobench-p | 94.2 | 86.9 | 7.7% | 49.34 | 50.17 | 53.9 | 50.1 | 7.0% | 49.25 | 48.46 |
| **Average** | **94.8** | **87.3** | **7.9%** | **43.21** | **43.04** | **53.7** | **49.6** | **7.6%** | **42.65** | **41.85** |
| *LongBenchV2 (by Seq. Len.)* | | | | | | | | | | |
| 8K – 16K | 89.1 | 82.4 | 7.5% | 37.04 | 39.74 | – | – | – | – | – |
| 16K – 32K | 137.8 | 126.1 | 8.5% | 32.58 | 35.94 | – | – | – | – | – |
| **Average** | **113.5** | **104.2** | **8.1%** | **34.81** | **37.84** | – | – | – | – | – |

Table 11: Comparison of KV Block Usage and Accuracy with and without Relocation for LongBench (LongChat-7B-v1.5-32K, 32K). Block size 16. 'No Reloc.': the KV tokens keep the chronological order in the same way they were generated during prefill. 'Reloc': the KV tokens are relocated after the prefill phase. 'Reduction': Percentage decrease in the critical block count from 'No Reloc' to 'Reloc'.

| | **Block Count** | | **Reduction** | **Accuracy** | |
|---|---|---|---|---|---|
| **Dataset** | **No Reloc.** | **Reloc.** | | **No Reloc.** | **Reloc.** |
| narrativeqa | 245.6 | 203.6 | 17.1% | 79.64 | 79.65 |
| qasper | 68.4 | 63.9 | 6.5% | 93.40 | 93.34 |
| multifieldqa_en | 85.5 | 76.6 | 10.4% | 92.01 | 91.76 |
| hotpotqa | 151.5 | 128.0 | 15.6% | 86.47 | 85.72 |
| 2wikimqa | 94.1 | 83.3 | 11.5% | 92.13 | 92.03 |
| musique | 175.0 | 147.4 | 15.7% | 83.86 | 82.84 |
| gov_report | 102.8 | 91.6 | 10.8% | 89.75 | 89.06 |
| qmsum | 140.4 | 119.0 | 15.2% | 86.34 | 86.00 |
| **Average** | **132.9** | **114.2** | **14.1%** | **87.95** | **87.55** |

Table 12: Comparison with Ada-KV on 12 LongBench datasets (LLaMA-3.1-8B-Instruct). Numbers in the headers are token budgets.

| Dataset | AdaKV-512 | AdaKV-1024 | Quest+DisSparse (520) | DisSparse (610) |
|---|---|---|---|---|
| Qasper | 31.06 | 36.35 | 46.60 | 45.60 |
| MF-en | 50.27 | 51.63 | 52.50 | 52.90 |
| HotpotQA | 53.90 | 54.97 | 54.80 | 55.10 |
| 2WikiMQA | 43.48 | 43.95 | 46.50 | 44.70 |
| Musique | 28.16 | 28.73 | 24.80 | 28.00 |
| GovRep. | 22.02 | 24.28 | 34.50 | 33.90 |
| QMSum | 23.28 | 24.16 | 25.40 | 25.20 |
| M-News | 21.49 | 23.23 | 27.40 | 27.20 |
| TriviaQA | 91.67 | 91.48 | 90.30 | 90.60 |
| PR-en | 99.50 | 100.00 | 99.50 | 99.50 |
| LCC | 60.37 | 62.83 | 63.30 | 63.00 |
| Repo. | 55.35 | 56.49 | 59.70 | 60.40 |
| **Avg.** | 48.38 | 49.84 | 52.11 | 52.18 |

Table 13: Maximum concurrent requests in colocated serving.

| Prompt length | Dense | DisSparse | Ratio |
|---|---|---|---|
| 8K | 14 | 26 | 1.86× |
| 16K | 8 | 14 | 1.75× |

# C Limitations and Future Work

DisSparse has several limitations that suggest avenues for future work.

**End-to-end performance at small batch sizes:** DisSparse is approximately 1.5× slower in TPOT than dense vLLM at batch sizes 1–4 (Figure 10). Our two-stream pipeline is incompatible with vLLM's single-stream CUDA Graph implementation [63], exposing per-kernel launch overhead, an engineering limitation rather than a fundamental one. **INT4 selection fidelity:** The INT4 metadata cache introduces a small overshoot of 5–6 extra blocks at $p \in [0.8, 0.95]$ while maintaining $\geq 97\%$ recall (Table 1). The overshoot is benign for accuracy but slightly inflates bandwidth. **Disaggregated-only evaluation:** We isolate the decode stage via disaggregated prefill/decode. Colocated results are in Appendix B.1; a full study of prefill interference is future work. **One-shot token relocation:** The relocation kernel uses prefill attention scores as a static estimate of token importance. For long generation runs where attention patterns drift, periodic re-relocation may improve effectiveness. MoE serving and composition with KV quantization, prefix caching, and speculative decoding are left for future work.

# D Algorithms

**Algorithm 3** Token-Level INT4 Scoring + Top-$p$

**procedure** TOKENTOPP($Q$, $K_{\text{INT4}}$, $P_{\text{tgt}}$, $B$)
**Phase 1 (scoring kernel): Per-token scores** (context split across thread blocks)
  $m \leftarrow -\infty,\ l \leftarrow 0$
  **for** each tile $j$ of $K_{\text{INT4}}$-cache **do**
    $S_j \leftarrow QK_j^\top/\sqrt{d}$ ▷ INT4 dequant; mask tokens outside the selector
    Store $S_j$ in the score buffer
    Update streaming $(m, l)$ with $S_j$
  **end for**
  Store this split's partials $(m, l)$
**Phase 2 (top-$p$ kernel): Histogram threshold** (one thread block per request and query head)
  $M, Z \leftarrow$ merge partials $(m, l)$ across splits ▷ global max and normalizer
  $H \leftarrow 0^B$
  **for** each token $t$ **do**
    $p_t \leftarrow \exp(S_t - M)/Z$ ▷ on the fly, no separate softmax pass
    $H[\text{bucket}(p_t)] \mathrel{+}= p_t$ ▷ bucket from the float bits of $p_t$
  **end for**
  $b \leftarrow \max\{i : \text{RevCumSum}(H)_i \geq P_{\text{tgt}}\}$
  $\tau \leftarrow$ lower edge of bucket $b$
  **return** tokens $t$ with $p_t \geq \tau$
**end procedure**

**Algorithm 4** Full Implementation of INT4 Top-$p$ Attention with Block Allocation

**procedure** TOPPATTNALLOC($Q$, $K_{\text{int4}}$, $s$, $z$, $P_{\text{tgt}}$, $K_{\text{cap}}$, $B$, $W$, $\mathcal{T}_{\text{fetch}}$)
**Phase 1: Block LSE computation & running Top-K**
  $m \leftarrow -\infty^{\text{BLOCK_M}},\ l \leftarrow 0^{\text{BLOCK_M}},\ T_{\text{buf}} \leftarrow \{-\infty\}^{K_{\text{cap}}}$
  **for** each tile $j \in [j_{\text{start}}, j_{\text{end}})$ **do**
    Load $K_j^{\text{int4}}$, dequantize: $K_j \leftarrow (\text{unpack}(K_j^{\text{int4}}) - z_j) \cdot s_j$
    $S_j \leftarrow \text{scale} \cdot (Q_{\text{even}} K_j^{\text{even}\top} + Q_{\text{odd}} K_j^{\text{odd}\top})$
    Apply masks: softcap, causal, sliding-window, ALiBi, QQ-bias
    Reshape $S_j \in \mathbb{R}^{\text{BLOCK_M} \times N_b \times \text{BLOCK_SIZE}}$, $N_b = \text{TILE}/\text{BLOCK}$
    $m_j \leftarrow \max_{\text{tile}} S_j,\ E_j \leftarrow \sum \exp_2(S_j - m_j)$ ▷ per-block sum
    $V_j \leftarrow m_j + \log_2(E_j)$ ▷ block LSE, $\in \mathbb{R}^{\text{BLOCK_M} \times N_b}$
    Store $V_j \rightarrow$ `output_attn_ptr`
    $m_{\text{new}} \leftarrow \max(m, m_j)$
    $l \leftarrow l \cdot \exp_2(m - m_{\text{new}}) + \sum_b E_j \cdot \exp_2(m_j - m_{\text{new}})$
    $m \leftarrow m_{\text{new}}$
    $T_{\text{buf}} \leftarrow \text{TopK}_{K_{\text{cap}}}(\text{sort}(T_{\text{buf}} \parallel V_j))$ ▷ merge & truncate
  **end for**
  **if** `decode_id`[seq] $\geq \mathcal{T}_{\text{fetch}}$ **then return** ▷ skip allocation
  **end if**
**Phase 2: Threshold $\tau$ via fast-path or LSE-domain histogram**
  $L_{\text{glb}} \leftarrow m + \log_2(l)$
  $P_{\text{topK}} \leftarrow \sum \exp_2(T_{\text{buf}} - L_{\text{glb}}),\ V_{\max} \leftarrow \min(T_{\text{buf}}) - L_{\text{glb}}$
  **if** $P_{\text{topK}} \geq P_{\text{tgt}}$ **then** ▷ Fast path
    $\tau \leftarrow V_{\max} - \epsilon + L_{\text{glb}}$
  **else** ▷ Slow path
    $V_{\min} \leftarrow V_{\max} - W$
    **for** each tile $j$ over all logical blocks **do**
      Reload $V_j$ from `output_attn_ptr`; $\tilde{V}_j \leftarrow V_j - L_{\text{glb}}$
      $p_j \leftarrow \exp_2(\tilde{V}_j)$
      $P_{\text{above}} \mathrel{+}= \sum p_j \cdot \mathbb{1}[\tilde{V}_j > V_{\max}]$
      $b_j \leftarrow \lfloor B \cdot \text{clip}((\tilde{V}_j - V_{\min})/W, 0, 1) \rfloor$
      $H[b_j] \mathrel{+}= p_j \cdot \mathbb{1}[\tilde{V}_j \in [V_{\min}, V_{\max}]]$
    **end for**
    $C \leftarrow \text{cumsum}(H);\ R \leftarrow \text{sum}(H) - C + H$ ▷ reverse cumsum
    $b^* \leftarrow \max\{b : R[b] \geq P_{\text{tgt}} - P_{\text{above}}\}$ (or $-\infty$ if none)
    $\tau \leftarrow V_{\min} + (b^*/B) \cdot W + L_{\text{glb}}$
  **end if**
**Phase 3: Block selection & free-pool allocation**
  **for** each tile $j$ over all logical blocks **do**
    Reload $V_j$; heavy $\leftarrow V_j \geq \tau$
    keep $\leftarrow$ heavy $\vee$ is_sink($j$) $\vee$ is_local($j$)
    need $\leftarrow \bigvee_{\text{queries}}$ keep ▷ reduce over BLOCK_M
    Load local & remote block-table entries
    claim $\leftarrow$ need $\wedge$ (local $< 0$) $\wedge$ (remote $\geq 0$)
    $n \leftarrow \sum$ claim; base $\leftarrow$ atomicAdd(free_pool_head, $n$)
    idx $\leftarrow$ base + cumsum(claim) $- 1$
    new_blk $\leftarrow$ free_pool[idx] where claimed
    Store new_blk $\rightarrow$ `new_block_ids_ptr`
  **end for**
**end procedure**

## 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: All the claims made in the abstract and introduction are empirically verified in the evaluation section 5.

   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: The paper discusses its potential limitations in Section C.

   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: This paper presents a systems contribution: an end-to-end integration of top-$p$ sparse attention into vLLM, including INT4 scoring and histogram top-$p$ kernels and a token relocation algorithm. The paper does not include formal theoretical results, theorems, or proofs. All claims are supported by empirical evaluation (Section 5).

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 paper provides detailed pseudocode for our contributed kernels: the INT4 scoring and histogram top-$p$ kernels (Algorithm 1, with the complete version in Appendix D) and the token relocation algorithm (Algorithm 2). Section 5 specifies the exact model checkpoints (Llama-3.1-8B-Instruct, LLaMA2-7B-Chat, Longchat-7B-v1.5-32k, Qwen2.5-7B-Instruct, Qwen3-32B, Phi-3-mini-128k-instruct, all from public Hugging Face releases), benchmark datasets (LongBench [12], LongBenchV2 [11], RULER [71], GSM8K [59], COQA [60], PG-19 [61]), hardware configuration, and hyperparameters including the top-$p$ threshold ($p = 0.95$ for LLaMA-2/3.1, $p = 0.85$ for LongChat), block size (16 tokens), $K_{\text{cap}} = 8$, and relocation parameters (sorting distance 4096, sort regions 3). All baselines (Quest, DS, Twilight) are configured following their official repositories, with the same skip-first-two-layers convention applied for fair comparison.

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 paper provides detailed algorithmic descriptions (Algorithm 1, Algorithm 2, and the full version in Appendix D) and complete experimental setup (Section 5), including model checkpoints (Llama-3.1-8B-Instruct, LLaMA2-7B-Chat, Longchat-7B-v1.5-32k, Qwen2.5-7B-Instruct, Qwen3-32B, Phi-3-mini-128k-instruct), benchmark datasets (LongBench, LongBenchV2, RULER, GSM8K, COQA, PG-19), hardware configuration, and hyperparameters. The code is available at `https://github.com/nnurlan008/DisSparse`.

   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: All related details are in Section 5.

   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: We report accuracy from single runs and do not report error bars, due to the cost of repeating each configuration. Results can differ slightly between runs and GPU types.

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: Accuracy and kernel experiments ran on NVIDIA A100 and RTX 6000 Ada GPUs, PG-19 perplexity on an RTX PRO 6000 Blackwell GPU, and system experiments on two A100-40GB GPUs (Section 5).

   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: The research conducted in the paper conforms with the NeurIPS Code of Ethics.

   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: [N/A]

    Justification: This paper does not directly have societal impacts as it only optimizes model serving efficiency. However, by reducing the cost of LLM inference, it could indirectly amplify both the positive and negative societal impacts of LLMs deployed at scale.

    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: This paper does not release data or models that have a high risk for misuse.

    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: The paper uses open-source repositories and publicly available models for the evaluation. (Section 5). All the use of the existing assets in this paper respects the original licensing.

    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: We release our code at `https://github.com/nnurlan008/DisSparse`, with documentation in the repository.

    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: This paper does not involve crowdsourcing nor research with human subjects.

    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: This paper does not involve crowdsourcing nor research with human subjects.

    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 core method and algorithm development of the paper does not involve LLM usage.

    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.