Step-dLLM: Adaptive Step-aware Sparse Attention for Efficient Diffusion LLM Inference

Zhichen Zeng[1]* **Xichong Zhang**[2]* **Junpan Wu**[3] **Yifei Zuo**[4] **Chi-Chih Chang**[5]
Jiayi Wang[1] **Maohua Nie**[1] **Ji Liu**[6] **Ang Li**[1] **Banghua Zhu**[1]

[1]University of Washington [2]University of Waterloo [3]University of Wisconsin, Madison
[4]Northwestern University [5]Cornell University [6]Meta AI

Abstract

Diffusion large language models (dLLMs) generate tokens in parallel via iterative denoising, but their bidirectional attention prevents the KV caching used by autoregressive models and recomputes the full attention map at every step, incurring quadratic per-step cost. Existing sparse attention methods for dLLMs construct a sparse mask after an initial full-attention warm-up and reuse it for the remaining steps, while allocating the sparsity budget uniformly across heads. We argue both choices ignore key structural properties of attention in dLLMs: empirically, attention patterns are only locally consistent across steps and shift abruptly at certain transitions, and different heads concentrate their attention mass to very different extents. To this end, we design **Step-dLLM**, which periodically refreshes a block-sparse mask at anchor steps along the denoising trajectory and reuses it within each locally consistent interval, while employing a head-adaptive global budget that more aggressively compresses heads with concentrated attention. Backed by a fused IO-aware Triton kernel for block-sparse attention, Step-dLLM applies to pre-trained LLaDA-1.5 and Dream-7B-Instruct without retraining, achieving a $2.78\times$ end-to-end speedup over dense FlashAttention on LLaDA-1.5 at 8K context with 256 denoising steps and 90% sparsity, while improving the accuracy–latency trade-off over prior sparse-attention methods.

1 Introduction

Recently, diffusion large language models (dLLMs) have achieved significant progress in the area of natural language processing [17, 38, 4, 1, 30, 21]. Traditional autoregressive language models (ARs), such as DeepSeek [9] and Qwen [18], rely on causal attention mechanisms, which restrict each token to attending only to its preceding context. In contrast, dLLMs leverage bidirectional context understanding and generate the entire sequence in parallel. By exploiting this capability, dLLMs achieve strong performance in language generation and represent a promising alternative to ARs.

Despite the advantages of parallel decoding, dLLMs still suffer from high generation latency. In AR models, KV caching [36, 13] allows each decoding step to reuse previously computed key–value pairs, reducing the per-step attention cost to linear in the sequence length. In contrast, dLLMs [17, 4, 30] rely on bidirectional attention, where all tokens, including those under denoising, attend to one another. Since token representations are updated at every denoising step, exact reuse of the standard autoregressive KV cache is unavailable without additional assumptions, and vanilla dLLMs recompute full-sequence attention repeatedly. As a result, dLLMs incur quadratic per-step cost across multiple denoising iterations, which limits their practical efficiency on long contexts [26, 24, 16].

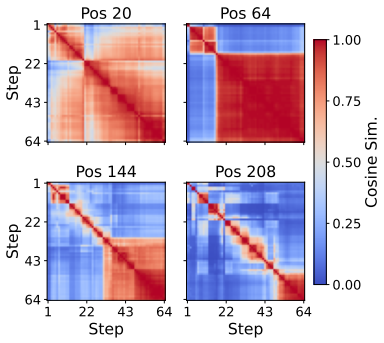
To improve dLLM efficiency, recent work has explored two complementary directions. The first is semi-autoregressive generation, exemplified by block diffusion [26], which partitions the output

*Equal Contribution.

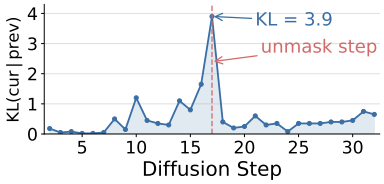
40th Conference on Neural Information Processing Systems (NeurIPS 2026).

into blocks, applies diffusion within each block, and decodes blocks autoregressively. This shortens each diffusion pass and enables KV caching for previous blocks. The second direction is sparse attention, which is orthogonal to block diffusion and reduces cost by attending only to salient token interactions within each step. This paper focuses on sparse attention. Although sparse attention has been widely studied in AR models [11, 27], those methods rely on structured causal patterns and do not directly transfer to dLLMs, where bidirectional attention evolves as tokens are progressively denoised. SparseD [25] takes a first step toward sparse attention for dLLMs that runs an initial fraction of denoising steps with full attention, constructs a sparse mask from these attention maps, and reuses it for the remaining steps. This exploits substantial cross-step similarity, but our measurements show that high similarity within a local interval does not guarantee that all salient connections persist across distant steps.

To verify the non-stationarity of attention patterns across denoising steps, we examine how the attention distribution of a query evolves along its denoising trajectory. Figure 1a visualizes the cosine similarity of attention patterns between every pair of denoising steps for four query positions $\{20, 64, 144, 208\}$ over 64 steps. The heatmaps reveal a consistent block-wise structure across different query locations: similarity remains high *within* contiguous step intervals but drops sharply at some specific steps. This indicates that attention patterns are *locally* stationary but globally non-stationary. Figure 1b further quantifies this behavior by plotting, for a representative query, the KL divergence between attention patterns at consecutive steps over 32 steps. The curve stays small for most steps but exhibits a dominant spike (KL$\approx$3.9) at the step where the query token transitions from masked to unmasked, marking an abrupt shift in its attention pattern. These observations challenge the implicit assumption behind static mask reuse, namely that the top-K important connections identified at early steps remain valid throughout all subsequent denoising steps. They instead motivate a strategy that reuses a sparse mask within each locally stationary interval for improved computational efficiency while refreshing it at the selected steps along the denoising trajectory.



(a) Cross-step cosine sim.



(b) KL across steps.

Figure 1: Attention pattern dynamics across denoising steps.

Motivated by these dynamics, we propose **Step-dLLM**, a sparse attention framework for dLLMs that refreshes the mask at selected anchor steps and reuses it only within locally consistent regions, avoiding per-step reconstruction. We further observe pronounced head-wise heterogeneity: heads differ substantially in how concentrated their attention mass is, so a uniform per-head budget is suboptimal. Step-dLLM therefore combines anchor-step mask refresh with head-adaptive global budget allocation, realized through fused Triton kernels. In summary, our contributions are as follows:

- We empirically identify two key properties of sparse attention in dLLMs: attention patterns are only *locally* consistent across denoising steps, and different heads exhibit substantially different concentration behaviors.
- Building on these observations, we propose **Step-dLLM**, which periodically refreshes sparse masks, reuses them across nearby denoising steps, allocates sparsity budgets adaptively across heads, and realizes the mask-refresh and mask-reuse stages with fused Triton kernels.
- We evaluate Step-dLLM on LLaDA-1.5 and Dream-7B-Instruct, achieving a $2.78\times$ end-to-end speedup over dense FlashAttention on LLaDA-1.5 at 8K context with 256 denoising steps and 90% sparsity, alongside improved accuracy–latency trade-offs over prior sparse-attention methods.

2 Background and Related Work

2.1 Preliminaries

Diffusion large language models generate text by iteratively unmasking a corrupted sequence over S denoising steps. Let $\mathcal{V}$ be the vocabulary and L the total sequence length, comprising m prompt tokens and $L - m$ response tokens at positions $\mathcal{R} = \{m+1, \ldots, L\}$. Starting from $x^{(0)}$ in which all response positions are set to $[\mathrm{MASK}]$, each step s applies a Transformer denoiser f_θ to produce per-token distributions $\pi_i^{(s)} = \mathrm{Softmax}\big(f_\theta(x^{(s-1)})_i\big)$ for $i \in \mathcal{R}$; a subset of currently masked positions is then unmasked by sampling from $\pi_i^{(s)}$ according to a decreasing mask-ratio schedule, while prompt tokens stay frozen. Because dLLMs predict *all* masked tokens simultaneously, f_θ uses **bidirectional** self-attention over the full length-L sequence at every step, incurring $\mathcal{O}(L^2)$ cost per step and precluding direct, exact reuse of the standard autoregressive KV cache. Reducing this per-step quadratic cost is the central goal of our work.

2.2 Related Work

Diffusion language models. Discrete diffusion was first developed in D3PM [2], SEDD [15], and MDLM [19], which together closed much of the gap with autoregressive models. LLaDA [17] scaled the paradigm to 8B parameters with subsequent multimodal, mixture-of-experts, and preference-optimized variants [33, 38, 37].

Efficient inference for dLLMs. Acceleration efforts follow two complementary directions: reducing the number of decoding passes via block-wise (semi-autoregressive) generation, which also enables KV caching across completed blocks [1, 26, 24]; and reducing per-step attention cost via sparse attention. The closest prior work, SparseD [25], uses full attention during an initial warm-up fraction, constructs a sparse mask from these attention maps, and reuses it for the remaining steps. We show that the step-dependent drift of dLLMs attention makes such one-shot reuse suboptimal under high sparsity, and address this with anchor-step mask refresh and head-adaptive budget allocation; our method is orthogonal to block-wise generation.

Sparse attention. Classical sparse attention uses fixed patterns [5, 3, 35], while recent dynamic methods construct block-sparse masks on the fly via online estimation, query-aware selection, or learned gating [11, 22, 39, 10, 34], complemented by IO-aware kernels [8, 7, 31, 29]. All of these target *causal* attention and rebuild the mask at every forward pass. Our work targets *bidirectional* dLLMs attention and amortizes mask construction across denoising steps by exploiting the temporal stability unique to iterative denoising (Section 4.1).

3 Analysis and Motivations

In this section, we analyze attention pattern dynamics during dLLMs inference and present two observations that motivate the design of Step-dLLM.

3.1 Cross-step attention similarity

Existing sparsity methods in dLLMs, such as SparseD [25], are built on the observation that attention patterns exhibit noticeable temporal consistency across inference steps. Based on this observation, they estimate the sparse attention pattern once and reuse it in subsequent denoising steps, thereby avoiding repeated pattern construction overhead. While this assumption is partially valid, we find that it is insufficient to fully characterize attention dynamics in dLLMs. For a query position i at denoising step s, we define its attention pattern as $\mathbf{a}_i^{(s)} = \mathrm{Softmax}\left(\frac{\mathbf{q}_i^{(s)}\mathbf{K}^{(s)\top}}{\sqrt{d}}\right)$, which represents the attention distribution of the query token over all key positions at step s.

Unlike autoregressive decoding, dLLMs inference proceeds through an iterative denoising process, in which the visible context and token certainty evolve continuously across steps. As more tokens become unmasked, the dependency structure of the sequence is gradually refined, and the attention pattern may be reorganized accordingly. In particular, some key positions that are not salient at early

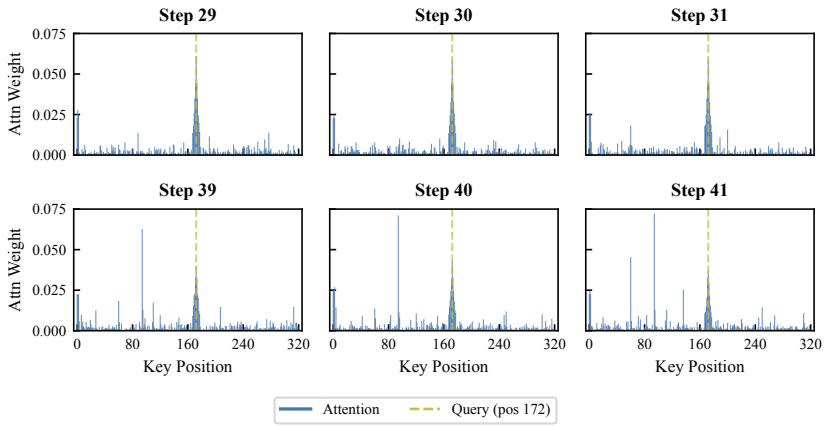
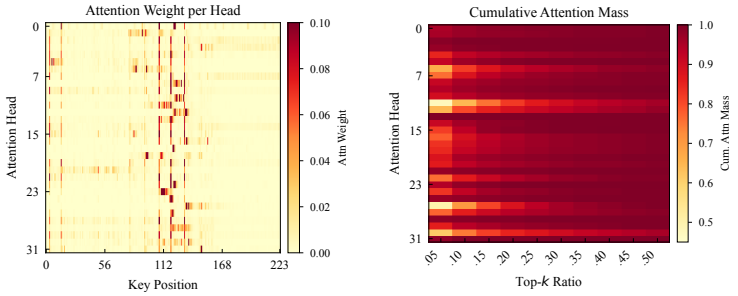


Figure 2: Attention pattern across different steps.

steps can become pivotal at later steps. As illustrated in Figure 2, which compares two groups of nearby denoising steps, the attention patterns are highly consistent within Steps 29–31 and within Steps 39–41, suggesting local reusability across nearby steps. However, comparing these two groups reveals a clear pattern shift: a new pivotal token at key position 95 emerges in Steps 39–41 but is not prominent in Steps 29–31. This indicates that, although the sparse attention pattern can be reused locally, it is not fully stationary throughout denoising.

As further shown in Figure 1b, the change between two consecutive attention patterns is measured by the KL divergence $D_{\mathrm{KL}}\left(\mathbf{a}_i^{(s)} \,\|\, \mathbf{a}_i^{(s-1)}\right) = \sum_{j=1}^{L} \mathbf{a}_{i,j}^{(s)} \log \frac{\mathbf{a}_{i,j}^{(s)}}{\mathbf{a}_{i,j}^{(s-1)}}$. Although many neighboring steps remain relatively similar, there exist several transition steps where the divergence increases sharply, suggesting abrupt changes in the underlying attention structure. This implies that directly reusing a sparse pattern estimated at early steps may lead to stale sparsity decisions: newly important attention links can be missed, while computation is still spent on entries that are no longer critical. These observations motivate us to refresh the sparse attention matrix at some selected anchor steps.



(a) Attention patterns of different heads for the same query token.

(b) Cumulative attention mass across heads.

Figure 3: Head-wise heterogeneity in sparse attention.

3.2 Head-wise Heterogeneity in Sparse Attention

Head specialization has been observed in BERT [6], in-context learning [32], and head-wise memory architectures [12]. These studies motivate treating heads individually; our analysis concerns their concentration behavior during dLLM denoising. Recent sparse dLLM methods, such as SparseD [25] and Sparse-dLLM [20], reduce inference cost by selecting important token interactions or evicting low-saliency cache entries at the token level. These approaches implicitly treat the token as the

primary unit of sparsification, assuming that the importance of a token is largely consistent across attention heads. We find that this assumption does not hold in dLLMs.

As shown in Figure 3, attention heads exhibit heterogeneity in two complementary aspects. First, for the same query token, different heads attend to substantially different sets of key positions: some focus sharply on a narrow region while others distribute attention more broadly or emphasize entirely different tokens (Figure 3(a)). Second, heads differ in how concentrated their attention mass is: some are highly concentrated, where a small fraction of top-scoring entries already retains most of the attention mass, whereas others are much more dispersed and require a substantially larger fraction to achieve the same coverage (Figure 3(b)). This heterogeneity is qualitatively different from the temporal patterns exploited by prior work: SparseD pre-computes sparse patterns once and reuses them across steps, while Sparse-dLLM leverages the stability of token saliency to evict cache entries. These temporal mechanisms do not directly address the head-wise budget allocation studied here.

Together, these observations reveal that the key challenge in sparse attention for dLLMs is not only *which* token interactions to preserve, but also *how much* computation budget each head should receive. This motivates us to allocate the sparse attention budget globally across heads rather than enforcing a uniform per-head ratio, allowing concentrated heads to be compressed more aggressively while reserving more capacity for dispersed ones.

4 Methodology

We present **Step-dLLM**, an efficient inference framework for dLLMs. It combines *cross-step mask reuse* to amortize sparse-pattern identification with a *head-adaptive budget* to allocate sparsity across attention heads. Both anchor and non-anchor steps are implemented with fused Triton kernels. Figure 4 provides an overview.

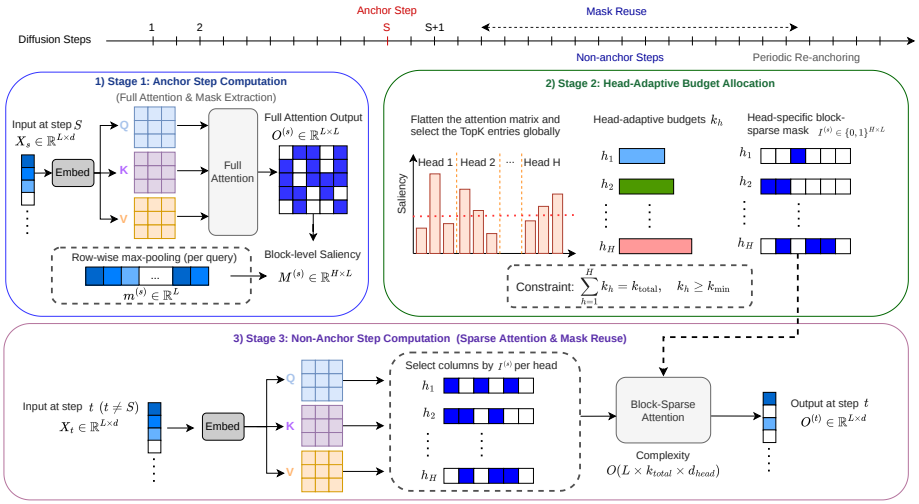


Figure 4: Overview of **Step-dLLM**. At an anchor step, the fused attention pass produces the dense output and block-level saliency (Stage 1). For each query block, saliency entries compete across all heads and key blocks for a shared top-K budget; a minimum allocation yields head-specific sparse masks (Stage 2). The next $\Delta - 1$ steps reuse only these masks, recomputing Q, K, V and the attention weights from the current hidden states (Stage 3). A single query block per head is illustrated.

4.1 Cross-Step Sparse Mask Reuse

As shown in Section 3, the attention patterns in dLLMs are locally consistent across nearby denoising steps but are not globally stationary: cumulative drift and occasional abrupt shifts make the one-shot mask increasingly stale as denoising progresses. This motivates our strategy: *reuse the sparse mask within several consecutive steps, but refresh it at some anchor steps* to prevent the accumulated mismatch from degrading generation quality.

Formally, let L denote the total sequence length, b the block size, and $N_b = \lceil L/b \rceil$ the number of blocks. We partition the S diffusion steps into consecutive groups of size Δ. Within each group, the first step serves as the *anchor* and the remaining $\Delta-1$ steps reuse the anchor's mask.

Anchor steps compute full attention and extract block-level saliency $\mathbf{M}$ in the same fused pass. Budget allocation and top-k selection then construct the mask $\mathcal{I}$ (Section 4.2):

$$\mathbf{M},\ \mathbf{O}^{(s)} = \texttt{Modified-Flash-Attn}(\mathbf{Q}^{(s)}, \mathbf{K}^{(s)}, \mathbf{V}^{(s)},\ b). \tag{1}$$

Non-anchor steps freeze the cached mask $\mathcal{I}$ and restrict attention to the selected blocks. Note that only the sparsity pattern is inherited; $\mathbf{Q}^{(s)}$, $\mathbf{K}^{(s)}$, and $\mathbf{V}^{(s)}$ are freshly computed from the current step's hidden states, so the attention weights remain faithful to the evolving token representations:

$$\mathbf{O}^{(s)} = \texttt{Block-Sparse-Attn}(\mathbf{Q}^{(s)}, \mathbf{K}^{(s)}, \mathbf{V}^{(s)},\ \mathcal{I}). \tag{2}$$

The mask is refreshed every Δ steps. Let k be the average number of retained key blocks per head and query block. Suppressing head count and feature dimension, the average attention cost is $\mathcal{O}(L^2/\Delta + Lkb)$, excluding mask-selection overhead. At fixed target sparsity, k scales with $(1-\rho)L/b$, so attention remains quadratic in L with a smaller coefficient. Algorithm 1 summarizes the procedure. The fixed schedule bounds refresh count without computing a drift signal at every step; Appendix C compares it with adaptive anchors.

Algorithm 1 Step-dLLM: Step-Aware Block-Sparse Attention with Head-Adaptive Budget

Require: Steps S, block size b, sparsity ρ, min budget $k_{\min}$, interval Δ, heads H
Ensure: $\mathbf{O}^{(s)}$ for $s = 1, \ldots, S$

$N_b \leftarrow \lceil L/b \rceil$; $K \leftarrow H \cdot \lfloor (1-\rho) N_b \rfloor$; $\mathcal{I} \leftarrow \varnothing$ ▷ #blocks; per-query-block budget; cached mask
for $s = 1$ **to** S **do**
  Compute $\mathbf{Q}^{(s)}, \mathbf{K}^{(s)}, \mathbf{V}^{(s)} \in \mathbb{R}^{H \times L \times d}$ from current hidden states
  if $\mathcal{I} = \varnothing$ **or** $s \equiv 1 \pmod{\Delta}$ **then** ▷ Anchor step
    $\mathbf{M}, \mathbf{O}^{(s)} \leftarrow \texttt{Modified-Flash-Attn}(\mathbf{Q}^{(s)}, \mathbf{K}^{(s)}, \mathbf{V}^{(s)}, b)$ ▷ $\mathbf{M} \in \mathbb{R}^{H \times N_b \times N_b}$
    for $p = 1$ **to** N_b **do**
      $\mathcal{T}_p \leftarrow \text{Top-}K(\{\mathbf{M}_{pq}^{(h)}\}_{h,q})$; $c_h(p) \leftarrow |\{(h', q) \in \mathcal{T}_p : h' = h\}|$ ▷ Global competition
      $k_{h,p} \leftarrow \max(c_h(p), k_{\min})$; $\mathcal{I}_p^{(h)} \leftarrow \text{Top-}k_{h,p}(\{\mathbf{M}_{pq}^{(h)}\}_q)$ ▷ Per-head budget & sparse mask
    end for
  else ▷ Non-anchor step
    $\mathbf{O}^{(s)} \leftarrow \texttt{Block-Sparse-Attn}(\mathbf{Q}^{(s)}, \mathbf{K}^{(s)}, \mathbf{V}^{(s)}, \mathcal{I})$ ▷ Reuse cached mask $\mathcal{I}$
  end if
end for

4.2 Block-Sparse Mask Construction with Head-Adaptive Budget

We now describe how the sparsity mask $\mathcal{I}$ is constructed at each anchor step. The procedure involves two stages: computing block-level saliency scores and allocating a non-uniform budget across heads.

Block-level saliency scoring. For each attention head h, query block p, and key block q, we compute a coarse saliency score via block-wise max-pooling over the attention logits:

$$\mathbf{M}_{pq}^{(h)} = \max_{(m,n) \in [b] \times [b]} \frac{(\mathbf{Q}_p^{(h)})_m (\mathbf{K}_q^{(h)})_n^\top}{\sqrt{d}}, \tag{3}$$

where $\mathbf{M}_{pq}^{(h)}$ captures the maximum attention affinity within each block pair for head h. This score is reduced from logits already computed in the full attention pass, avoiding a second read of Q, K for saliency estimation; storing the block scores and selecting the mask still incur overhead.

Head-adaptive budget allocation. As shown in Section 3, different heads exhibit distinct concentration behaviors: some heads focus sharply on a few key blocks while others distribute attention broadly. A uniform per-head budget cannot accommodate this heterogeneity. We therefore determine each head's sparsity budget through a global competition over saliency scores.

For each query block p, we flatten the saliency scores $\{\mathbf{M}_{pq}^{(h)}\}$ across all heads into a single vector of size $H \cdot N_b$ and select the top-K entries, where $K = H \cdot \lfloor (1-\rho) \cdot N_b \rfloor$ is the budget derived from the target sparsity ratio ρ. Let $c_h(p)$ denote the number of entries head h wins at query block p. The per-head budget for query p is:

$$k_{h,p} = \max(c_h(p),\ k_{\min})\,, \tag{4}$$

which ensures every head retains at least $k_{\min}$ blocks for that query block. Each head then independently constructs its sparse mask:

$$\mathcal{I}_p^{(h)} = \text{Top-k}_{\text{h,p}}\Big(\{\mathbf{M}_{pq}^{(h)}\}_{q=1}^{N_b}\Big)\,, \quad \mathcal{I}^{(h)} = \{\mathcal{I}_p^{(h)}\}_{p=1}^{N_b}. \tag{5}$$

The per-head allocation emerges from competition over saliency rather than an explicit dispersion metric. Because the scores are raw logits, their scales may differ across heads; this heuristic does not guarantee an allocation ordered by normalized attention concentration. Cross-head score calibration remains a limitation.

5 Experiments

5.1 Experimental Setup

Setup. We apply Step-dLLM to two representative dLLMs, LLaDA-1.5 [37] and Dream-7B-Instruct [30], and evaluate on five benchmarks covering reasoning (GSM8K, MATH), code (HumanEval), and long-context retrieval (NIAH, RULER). We compare against the dense backbone, the AR-style sparse method StreamingLLM [27], and the dLLM-targeted sparse method SparseD [25]. We report per-benchmark accuracy for quality, and kernel-level and end-to-end speedup over dense Flash Attention [8] for efficiency.

Implementation Details. All experiments are conducted on a single NVIDIA H100 GPU. For our proposed Step-dLLM, we largely follow the hyperparameter conventions of SparseD [25]. We set the attention block size to $b = 16$ and the semi-autoregressive generation block length to $B = 32$ for all datasets; the latter controls the ratio between autoregressive steps and diffusion steps and is distinct from the attention block size b. The re-anchor interval is set to $\Delta = 0.2 \cdot S$, i.e., the sparse attention mask is refreshed every $0.2 \cdot S$ denoising steps, corresponding to a refresh interval ratio of $f = \Delta/S = 0.2$. All reported experiments use $k_{\min} = 1$, i.e., at least one key block per head and query block (not a fixed percentage). This floor can increase the realized keep rate above $1-\rho$; we report nominal sparsity and do not claim exact budget preservation after flooring. The global budget K_{total} is derived from the target sparsity ratio ρ via $K_{\text{total}} = H \cdot \lfloor (1-\rho) \cdot N_b \rfloor$ (Section 4.2). For all benchmarks, we evaluate under four sparsity ratios $\rho \in \{50\%, 80\%, 90\%, 95\%\}$. SparseD uses the same (b, B, ρ) configuration, with its early-step full-attention skip ratio set to 5%. For StreamingLLM, we fix the proportion of sink tokens to $s_0 = 10\%$ of the total generation length and derive the sliding window size W accordingly. Model, dataset, and baseline descriptions are provided in Appendix F.

5.2 Accuracy

Table 1 reports accuracy on LLaDA-1.5 and Dream-7B-Instruct under four sparsity ratios. For NIAH and RULER, we report the average over their sub-tasks; per-sub-task scores are in Appendix A. Step-dLLM improves accuracy over SparseD in most settings, especially on reasoning and code tasks at high sparsity; the ordering on long-context tasks depends on the backbone and sub-task.

Reasoning and code generation. On GSM8K, MATH, and HumanEval, both sparse methods largely retain the dense baseline on LLaDA-1.5 at $\rho = 50\%$; Dream-7B-Instruct still exhibits a GSM8K gap to dense. The gap opens at higher sparsity. At $\rho = 90\%$ on GSM8K, Step-dLLM increases accuracy from 29.64% to 61.79% on LLaDA-1.5 and from 22.21% to 32.68% on Dream-7B-Instruct over SparseD. At $\rho = 95\%$, SparseD collapses to single-digit accuracy (9.86% and 4.40%). Step-dLLM retains 18.80% and 15.31%. The same trend holds on MATH and HumanEval. This is because the one-shot mask in SparseD grows progressively stale as denoising proceeds. Periodic refresh re-aligns the sparse pattern with the evolving attention, which prevents the late-step collapse that SparseD suffers under tight budgets.

Table 1: Main results on LLaDA-1.5 and Dream-7B-Instruct (accuracy in %). NIAH and RULER columns are averaged across their sub-tasks; per-sub-task scores are in Appendix A. Within each sparsity block, **bold** marks the highest score per (backbone, metric). Step-dLLM rows are highlighted.

		LLaDA-1.5					Dream-7B-Instruct				
Sparsity	Method	GSM8K	Math	HEval	NIAH	RULER	GSM8K	Math	HEval	NIAH	RULER
Dense		81.04	37.20	40.24	100.00	76.78	77.41	45.60	59.12	95.06	81.33
50%	StreamingLLM	74.07	38.20	39.02	18.75	48.77	38.13	8.80	42.70	10.16	41.77
	SparseD	80.59	37.40	**42.68**	**100.00**	74.86	70.50	43.60	56.10	94.78	**80.44**
	Step-dLLM	**81.65**	**40.00**	**42.68**	**100.00**	**75.05**	**71.80**	**44.00**	**58.54**	**94.81**	79.75
80%	StreamingLLM	19.11	13.60	19.51	2.50	25.75	19.56	10.40	25.00	1.84	9.60
	SparseD	53.68	29.20	36.59	88.79	71.75	49.81	37.80	53.66	77.72	75.89
	Step-dLLM	**77.56**	**36.40**	**38.41**	**99.74**	**73.73**	**68.46**	**38.60**	**54.27**	**85.59**	**76.15**
90%	StreamingLLM	0.08	0.00	2.43	0.00	0.48	0.00	0.00	2.44	0.00	0.00
	SparseD	29.64	20.00	30.49	89.63	65.82	22.21	28.80	43.90	**77.06**	70.90
	Step-dLLM	**61.79**	**26.20**	**34.76**	**99.14**	**70.97**	**32.68**	**32.00**	**50.00**	76.12	**72.35**
95%*	SparseD	9.86	9.40	20.12	90.06	64.23	4.40	14.80	31.10	**74.60**	64.47
	Step-dLLM	**18.80**	**16.60**	**23.17**	**96.84**	**66.78**	**15.31**	**19.20**	**35.98**	72.36	**66.22**

* StreamingLLM is omitted at $\rho = 95\%$ as all metrics collapse to zero on both backbones.

Long-context retrieval. On NIAH, Step-dLLM is essentially lossless up to $\rho = 90\%$ on LLaDA-1.5, increasing from 89.63% for SparseD to 99.14%, and still retains 96.84% at $\rho = 95\%$. On Dream-7B-Instruct, the NIAH gain holds at $\rho = 80\%$, but SparseD is better at 90% and 95% (77.06 vs. 76.12 and 74.60 vs. 72.36). On RULER, Step-dLLM leads on LLaDA-1.5 at all evaluated sparsities and on Dream-7B-Instruct except at 50% (e.g., from 65.82% to 70.97% on LLaDA-1.5 at $\rho = 90\%$). Appendix A shows the gain concentrates on multi-needle and compositional sub-tasks. These sub-tasks depend on a few sparsely scattered key positions, so any mask staleness is amplified. As a result, periodic refresh together with head-adaptive budget allocation matters most exactly here.

In contrast, StreamingLLM collapses at high sparsity. On LLaDA-1.5 GSM8K, it drops from 74.07% at ρ=50% to nearly zero at ρ=90%, and we therefore omit its ρ=95% row. The reason is that StreamingLLM relies on causal-decoder priors (sink tokens plus a sliding window). These priors do not capture the bidirectional, step-evolving attention of dLLMs.

5.3 Efficiency

Kernel-level efficiency. We implement two fused Triton [23] kernels that save IO and memory in the spirit of Flash Attention [8, 7]. `Modified-Flash-Attn` extends Flash Attention [8] to produce both $\mathbf{O}$ and the block-level saliency matrix $\mathbf{M}^{(h)}$ in a single pass. `Block-Sparse-Attn` consumes the cached indices $\mathcal{I}^{(h)}$ and loads only the $k_{h,p}$ selected key/value blocks per query block. The anchor pass avoids materializing a token-level $L \times L$ attention map, but stores a block-level saliency tensor with HN_b^2 entries; at fixed b, this tensor still scales quadratically with L. As shown in Figure 5, the block-sparse kernel achieves $8.55\times$ over dense Flash Attention at 16K under $\rho = 95\%$ and around $2\times$ at $\rho = 80\%$ across all tested lengths. The reason is that the fused kernels turn algorithmic block-sparsity into real SRAM and HBM savings. At 50% sparsity the kernel instead reaches only $0.83\times$ at 1K and 2K: irregular K/V gathers and variable per-tile work can outweigh the reduced FLOPs. Kernel timings do not include the amortized anchor and mask-selection costs.

End-to-end efficiency. We further measure end-to-end latency of LLaDA-1.5-8B at 8K context across $S \in \{32, 64, 128, 256\}$. As shown in Figure 6, Step-dLLM beats SparseD at every (ρ, S) pair. At S=256, it reaches $1.42\times$, $1.75\times$, and $2.78\times$ over FlashAttention at $\rho = 75\%, 80\%, 90\%$, and the speedup grows with both S and ρ. The reason is that our fused IO-aware kernel skips unselected key/value blocks rather than masking them after a dense load. These measurements include dense anchor steps. Their speedups depend on context length, sparsity, refresh interval, and the fraction of runtime spent outside attention; kernel-level gains do not imply proportional generation speedups.

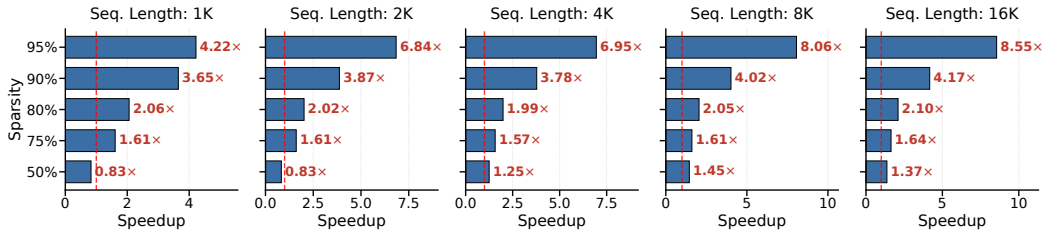


Figure 5: Kernel-level wall-clock speedup of the fused block-sparse attention kernel over the dense Flash Attention baseline.

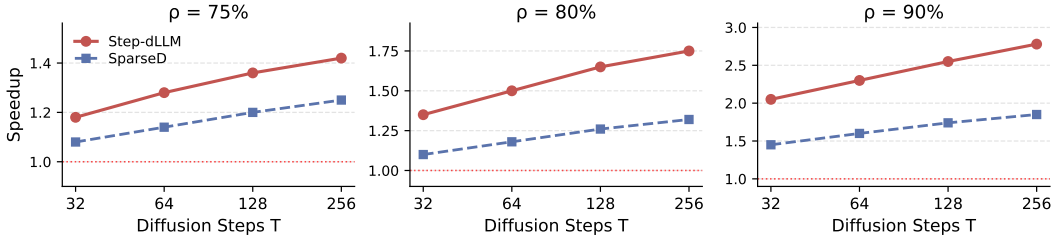


Figure 6: End-to-end speedup of LLaDA-1.5-8B over dense FlashAttention at 8K context.

Joint accuracy–latency trade-off. Appendix B reports the complete GSM8K sweep at approximately 1K sequence length, including $\rho = 50\%$. Dense reaches 81.04% at 50.39 ms/step. Step-dLLM at $\rho = 70\%$ reaches 80.74% at 48.50 ms/step, compared with SparseD's 80.59% at 59.94 ms/step at $\rho = 50\%$: a $1.24\times$ latency advantage at similar accuracy. At $\rho = 50\%$, Step-dLLM takes 51.23 ms/step, slightly slower than dense. Thus the practical gain over dense here is modest ($1.04\times$ at 70%), distinct from the larger 8K-context speedups. Small accuracy gains over dense may reflect evaluation variability or pruning of weak connections; they do not establish a denoising mechanism or statistical significance.

Implementation backends and contribution. Both sparse paths use fused Triton kernels: SparseD uses PyTorch `flex_attention`, compiled by TorchInductor, whereas Step-dLLM adapts SeerAttention's block-sparse kernel to heterogeneous per-head budgets. SparseD's mask construction operates on materialized attention maps and computes the dense output separately; our anchor pass combines dense output and block-saliency extraction. The accuracy contribution is the refresh and budget algorithm; efficiency reflects its co-design with the kernels, rather than the use of Triton alone. This is a comparison of complete implementations, not an isolation of algorithmic speed under an identical backend.

5.4 Ablation Studies

We ablate the two design choices in Step-dLLM, periodic mask refresh and head-adaptive budget allocation, and study sensitivity to the block size. Refresh and block-size ablations are conducted on LLaDA-1.5. Tables 2 and 3 include $\rho = 50\%$; Appendix D compares saliency pooling; Appendix E evaluates composition with Fast-dLLM dual caching.

Refresh frequency. Table 2 reports performance under varying refresh intervals f. On GSM8K at $\rho = 90\%$, SparseD increases from 29.64% to 43.90% between $f = 1.0$ and $f = 0.2$. Step-dLLM increases from 35.18% to 61.79% over the same interval. Refresh re-aligns the mask with the current attention pattern. At $\rho = 50\%$, neither method consistently dominates: GSM8K is 80.59 vs. 77.48 at $f = 1.0$, but 80.97 vs. 81.65 at $f = 0.2$ (SparseD vs. Step-dLLM). Thus the advantage is most pronounced under tight budgets.

Head-adaptive budget. Head-adaptive allocation contributes independently of refresh. At $f = 1.0$, neither method refreshes its mask; the head-adaptive versus uniform allocation is the only algorithmic difference in this ablation. On GSM8K at $\rho = 90\%$ it increases from 29.64% to 35.18%, and on

Table 2: Ablation on mask refresh interval. $f = \Delta/S$; lower f means more frequent refresh. For $f < 1$, SparseD denotes its refresh-augmented, uniform-budget variant. Accuracy in % on LLaDA-1.5, including 50% sparsity. G8K, VT denote GSM8K and RULER-VT; bold marks the better method within each configuration (ties included).

Interval	Method	Sparsity 50%			Sparsity 80%			Sparsity 90%			Sparsity 95%		
		G8K	MATH	VT	G8K	MATH	VT	G8K	MATH	VT	G8K	MATH	VT
f=1.0	SparseD	**80.59**	37.40	99.88	53.68	29.20	94.32	29.64	**23.00**	93.64	9.48	12.00	86.16
	Step-dLLM	77.48	**40.20**	**100.00**	**57.77**	**31.00**	**98.52**	**35.18**	21.20	**94.00**	**16.83**	**15.40**	**92.32**
f=0.5	SparseD	**81.50**	**40.60**	**100.00**	64.37	31.20	95.12	29.72	22.20	**97.04**	6.29	13.80	**95.40**
	Step-dLLM	81.27	**40.60**	**100.00**	**68.23**	**32.00**	**96.64**	**40.64**	**22.60**	96.56	**16.98**	**16.00**	95.36
f=0.3	SparseD	82.03	**40.80**	**100.00**	73.09	32.40	95.23	34.19	23.20	**97.96**	5.53	14.20	**97.04**
	Step-dLLM	**82.34**	**40.80**	**100.00**	**75.01**	**33.60**	**97.28**	**52.24**	**23.60**	97.04	**17.43**	**15.40**	96.36
f=0.2	SparseD	80.97	**40.60**	**100.00**	76.80	34.00	95.80	43.90	24.80	**97.96**	6.29	13.60	96.64
	Step-dLLM	**81.65**	40.00	99.60	**77.56**	**36.40**	**99.88**	**61.79**	**26.20**	97.50	**18.80**	**16.60**	**97.20**

RULER-VT at $\rho = 95\%$ from 86.16% to 92.32%. The reason is that a uniform per-head budget over-allocates blocks to heads with concentrated mass and under-allocates blocks to dispersed heads. Redistributing toward dispersed heads yields a better mask before any refresh is introduced.

Table 3: Ablation on attention block size b. Accuracy in % on LLaDA-1.5, including 50% sparsity. G8K, HE denote GSM8K and HumanEval; bold marks the better method within each configuration (ties included).

Block	Method	Sparsity 50%			Sparsity 80%			Sparsity 90%			Sparsity 95%		
		G8K	MATH	HE	G8K	MATH	HE	G8K	MATH	HE	G8K	MATH	HE
b=16	SparseD	80.59	37.40	**42.68**	53.68	29.20	36.59	29.64	20.00	30.49	9.86	9.40	20.12
	Step-dLLM	**81.65**	**40.00**	**42.68**	**77.56**	**36.40**	**38.41**	**61.79**	**26.20**	**34.76**	**18.80**	**16.60**	**23.17**
b=32	SparseD	**81.73**	34.80	**38.41**	50.87	**31.60**	37.80	17.51	21.60	28.66	8.64	10.60	12.80
	Step-dLLM	81.58	**41.00**	37.80	**63.23**	31.20	**39.12**	**39.50**	**26.00**	**34.15**	**19.18**	**16.20**	**23.78**

Block size. Table 3 compares $b = 16$ and $b = 32$. Step-dLLM improves GSM8K at both block sizes; the absolute margin depends on sparsity. On GSM8K at $\rho = 90\%$, Step-dLLM increases accuracy from 29.64% to 61.79% at $b = 16$ and from 17.51% to 39.50% at $b = 32$ over SparseD. At 50%, gains are mixed: Step-dLLM improves MATH at both block sizes, while SparseD slightly leads GSM8K and HumanEval at $b = 32$. Larger blocks remove more tokens per selection, motivating a joint choice of block size and sparsity.

6 Conclusion and Limitations

We presented **Step-dLLM**, a training-free sparse attention framework for diffusion LLMs. Motivated by local cross-step consistency and head-wise concentration heterogeneity, Step-dLLM refreshes a block-sparse mask at periodic anchor steps and uses a head-adaptive global budget, backed by a fused IO-aware Triton kernel. Experiments on LLaDA-1.5 and Dream-7B-Instruct demonstrate improved accuracy–latency trade-offs over prior sparse-attention methods. On LLaDA-1.5 at 8K context with 256 denoising steps and 90% sparsity, Step-dLLM achieves a $2.78\times$ end-to-end speedup over dense FlashAttention.

Limitations. Step-dLLM uses a heuristic anchor schedule rather than drift-triggered adaptive anchors, and is evaluated only on text-only dLLMs. The joint accuracy–latency sweep covers one model, task, and short-context setting; wider operating-point validation is needed. Raw-logit saliency may be affected by head-wise scale differences, and the minimum-head floor can relax nominal sparsity. Adaptive triggers and extension to multimodal diffusion models remain future work.

References

[1] Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025.

[2] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, pages 17981–17993, 2021.

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

[4] Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. LLaDA 2.0: Scaling up diffusion language models to 100B. *arXiv preprint arXiv:2512.15745*, 2025.

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

[6] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, 2019. URL `https://aclanthology.org/W19-4828/`.

[7] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024.

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

[9] DeepSeek-AI. DeepSeek-V3.2: Efficient Reasoning & Agentic AI. `https://huggingface.co/deepseek-ai/DeepSeek-V3.2`, 2025. Accessed: 2026-05-02.

[10] Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. SeerAttention: Learning intrinsic sparse attention in your LLMs. *arXiv preprint arXiv:2410.13276*, 2024.

[11] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems*, volume 37, pages 52481–52515, 2024.

[12] Taero Kim, Hoyoon Byun, Youngjun Choi, Sungrae Park, and Kyungwoo Song. MIDUS: Memory-infused depth up-scaling. *arXiv preprint arXiv:2512.13751*, 2025.

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

[14] Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

[15] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning*, 2024.

[16] Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, Xinyuan Zhang, Zhen Tao, Haibo Feng, Ziyun Jiang, Ying Xu, Zenan Huang, Yihong Zhuang, Haokai Xu, Jiaqi Hu, Zhenzhong Lan, Junbo Zhao, Jianguo Li, and Da Zheng. dInfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666*, 2025.

[17] Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

[18] Qwen Team. Qwen3.6-Max-Preview: Smarter, Sharper, Still Evolving. `https://qwen.ai/blog?id=qwen3.6-max-preview`, April 2026. Accessed: 2026-05-02.

[19] Subham Sekhar Sahoo, Marianne Arriola, Aaron Gokaslan, Edgar Mariano Marroquin, Alexander M Rush, Yair Schiff, Justin T Chiu, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[20] Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. Sparse-dLLM: Accelerating diffusion LLMs with dynamic cache eviction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 33038–33046, 2026.

[21] Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Yonghui Wu, and Hao Zhou. Seed Diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

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

[23] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. 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*, pages 10–19, 2019.

[24] Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion LLMs can do faster-than-AR inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*, 2025.

[25] Zeqing Wang, Gongfan Fang, Xinyin Ma, Xingyi Yang, and Xinchao Wang. SparseD: Sparse attention for diffusion language models. *arXiv preprint arXiv:2509.24014*, 2025.

[26] Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration for diffusion large language models. *arXiv preprint arXiv:2505.22618*, 2025.

[27] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=NG7sS51zVF`.

[28] Fangyuan Xu, Tanya Goyal, and Eunsol Choi. RefreshKV: Updating small KV cache during long-form generation. *arXiv preprint arXiv:2411.05787*, 2024.

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

[30] Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream-7B: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

[31] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. FlashInfer: Efficient and customizable attention engine for LLM inference serving. *arXiv preprint arXiv:2501.01005*, 2025.

[32] Kayo Yin and Jacob Steinhardt. Which attention heads matter for in-context learning? In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pages 72428–72461, 2025. URL `https://proceedings.mlr.press/v267/yin25e.html`.

[33] Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. LLaDA-V: Large language diffusion models for multimodal understanding. *arXiv preprint arXiv:2505.16933*, 2025.

[34] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

[35] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big Bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33, 2020.

[36] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023.

[37] Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. LLaDA 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

[38] Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, Hongrui Guo, Jiaqi Hu, Wentao Ye, Tieyuan Chen, Chenchen Li, Chengfu Tang, Haibo Feng, Jun Hu, Jun Zhou, Xiaolu Zhang, Zhenzhong Lan, Junbo Zhao, Da Zheng, Chongxuan Li, Jianguo Li, and Ji-Rong Wen. LLaDA-MoE: A sparse MoE diffusion language model. *arXiv preprint arXiv:2509.24389*, 2025.

[39] Kan Zhu, Tian Tang, Qinyu Xu, Yile Gu, Zhichen Zeng, Rohan Kadekodi, Liangyu Zhao, Ang Li, Arvind Krishnamurthy, and Baris Kasikci. Tactic: Adaptive sparse attention with clustering and distribution fitting for long-context llms. *arXiv preprint arXiv:2502.12216*, 2025.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction state our contributions, which are supported by the experiments in 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: We discuss limitations in Section 6.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

Justification: The paper is empirical and contains no theoretical results.

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: Algorithm 1, Section 5, and Appendix F describe the method, principal hyperparameters, and baseline configurations used to reproduce the main results.

   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: [No]

Justification: We do not provide a public code release with this paper; the evaluated datasets are public.

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: Section 5 specifies the experimental settings and baseline configurations; Appendix F provides additional model and benchmark descriptions.

   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 point estimates on standard benchmark splits, following the baseline evaluation protocol.

   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: All experiments run on a single NVIDIA H100 GPU, as stated in Section 5. Efficiency measurements are reported in Section 5 and Appendix B.

   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: We use only public models and benchmarks; no human subjects or sensitive data are involved.

   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 work concerns inference efficiency; societal impact is tied to the underlying models, not to accelerating them.

    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: We do not release any new models or scraped datasets.

    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: All models, baselines, and benchmarks are properly cited and used under their original licenses.

    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: [N/A]

    Justification: No new assets are released with this submission.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or 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: No human subjects are involved.

    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: LLMs are the subject of acceleration, not a component of the methodology; any LLM use is limited to writing and editing.

    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.

A Detailed Per-Sub-Task Results

This appendix lists the per-sub-task scores for the long-context benchmarks summarized in the main paper (Table 1). For each backbone, we separate the eight NIAH sub-tasks from the five RULER sub-tasks so that each table can be read at a normal font size. The "NIAH (avg)" and "RULER (avg)" columns in the main paper are obtained by averaging the values listed here, weighting all sub-tasks equally.

NIAH sub-tasks. The Needle-in-a-Haystack benchmark probes how reliably a model can locate a small piece of information that has been planted inside a long context. We evaluate eight variants. *Single-needle* tasks (**S_1, S_2, S_3**) ask the model to retrieve a single answer planted at one position; the three variants differ in needle length and document complexity. *Multi-key* tasks (**MK_1, MK_2, MK_3**) plant several distractor keys alongside the target so the model must disambiguate among multiple candidates. *Multi-query* (**MQ**) requires the model to answer several queries against the same context in one pass, while *multi-value* (**MV**) requires extracting several values associated with the same key. Together these settings cover both retrieval accuracy and the ability to track multiple pieces of evidence simultaneously.

RULER sub-tasks. RULER stresses long-context reasoning beyond simple retrieval. **VT** (variable tracking) asks the model to follow chains of variable assignments scattered across the context. **CWE** and **FWE** (common- and frequent-word extraction) probe whether the model can identify globally distinguished tokens by aggregating statistics over the entire input. **QS** and **QH** are single-document and multi-document QA, with the latter requiring the model to combine evidence from several passages. These sub-tasks therefore stress different aspects of long-context attention than NIAH: tracking, aggregation, and multi-hop composition.

Reading the tables. Within each sparsity block, the highest score per sub-task is marked in bold; Step-dLLM rows are shaded. The Dense row is provided as a non-sparse reference and is never bolded. Tables 4 and 5 cover LLaDA-1.5; Tables 6 and 7 cover Dream-7B-Instruct. A few patterns are worth noting beyond the main paper. First, the gap between Step-dLLM and SparseD is concentrated on the harder multi-needle and multi-value tasks (S_3, MK_3, MV); on the easier single-needle tasks both methods stay near the dense baseline up to high sparsity. Second, on RULER the relative ordering between methods varies more across sub-tasks: Step-dLLM dominates on tracking-heavy tasks (VT, CWE) where the salient key positions shift across denoising steps, while SparseD remains competitive on QA-style tasks where the relevant context tends to be more localized.

Table 4: Per-sub-task NIAH results on LLaDA-1.5 (accuracy in %). The eight sub-tasks are single-needle (S_1, S_2, S_3), multi-key (MK_1, MK_2, MK_3), multi-query (MQ), and multi-value (MV). Within each sparsity block, **bold** marks the highest score per sub-task. Step-dLLM rows are highlighted.

Sparsity	Method	S_1	S_2	S_3	MK_1	MK_2	MK_3	MQ	MV
Dense		100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
50%	StreamingLLM	20.80	20.20	18.40	21.20	20.40	15.40	19.20	14.40
	SparseD	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
	Step-dLLM	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
80%	StreamingLLM	5.00	2.40	1.40	3.40	3.60	0.00	2.10	2.10
	SparseD	**100.00**	**100.00**	71.20	**100.00**	**100.00**	74.20	85.85	79.10
	Step-dLLM	**100.00**	**100.00**	**100.00**	**100.00**	99.80	**99.60**	**99.90**	**98.60**
90%	StreamingLLM	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	SparseD	**100.00**	**100.00**	76.60	**100.00**	99.80	77.20	82.85	80.60
	Step-dLLM	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**97.60**	**99.65**	**95.90**
95%	SparseD	**100.00**	**100.00**	79.40	**100.00**	**100.00**	78.20	81.80	81.10
	Step-dLLM	**100.00**	**100.00**	**98.80**	**100.00**	**100.00**	**88.40**	**95.25**	**92.30**

Table 5: Per-sub-task RULER results on LLaDA-1.5 (accuracy in %). The five sub-tasks are variable tracking (VT), common-word extraction (CWE), frequent-word extraction (FWE), single QA (QS), and hard QA (QH). Within each sparsity block, **bold** marks the highest score per sub-task. Step-dLLM rows are highlighted.

Sparsity	Method	VT	CWE	FWE	QS	QH
Dense		100.00	52.40	64.20	91.30	76.00
50%	StreamingLLM	30.60	13.40	**73.80**	81.65	44.40
	SparseD	**99.88**	52.26	61.40	**83.17**	77.60
	Step-dLLM	99.60	**53.16**	60.87	83.03	**78.60**
80%	StreamingLLM	6.56	3.02	**76.27**	20.48	22.40
	SparseD	94.28	**57.80**	45.40	83.48	77.80
	Step-dLLM	**98.52**	56.46	51.73	**83.73**	**78.20**
90%	StreamingLLM	0.00	2.12	0.00	0.27	0.00
	SparseD	93.32	34.56	42.07	83.13	76.00
	Step-dLLM	**94.00**	**54.16**	**46.80**	**83.48**	**76.40**
95%	SparseD	86.64	39.92	39.00	**81.97**	**73.60**
	Step-dLLM	**92.32**	**44.76**	**44.07**	81.37	71.40

Table 6: Per-sub-task NIAH results on Dream-7B-Instruct (accuracy in %). Within each sparsity block, **bold** marks the highest score per sub-task. Step-dLLM rows are highlighted.

Sparsity	Method	S_1	S_2	S_3	MK_1	MK_2	MK_3	MQ	MV
Dense		100.00	100.00	91.00	98.00	99.80	77.60	97.70	96.40
50%	StreamingLLM	10.00	11.00	9.00	11.00	8.80	12.40	8.75	10.30
	SparseD	**100.00**	**100.00**	90.20	**98.00**	99.40	**75.80**	**97.45**	97.40
	Step-dLLM	**100.00**	**100.00**	**90.50**	97.80	**99.80**	75.60	97.25	**97.50**
80%	StreamingLLM	1.80	2.80	1.40	3.20	1.20	0.00	1.95	2.40
	SparseD	**100.00**	**99.80**	47.20	**98.80**	**99.80**	46.80	60.05	69.30
	Step-dLLM	**100.00**	**99.80**	**69.00**	98.20	**99.80**	**65.40**	**74.10**	**78.40**
90%	StreamingLLM	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	SparseD	**100.00**	**100.00**	44.60	**98.80**	**99.80**	**44.80**	**59.45**	**69.00**
	Step-dLLM	**100.00**	99.80	**47.00**	97.20	**99.80**	39.80	57.30	68.05
95%	SparseD	**100.00**	**100.00**	**40.80**	**98.40**	99.40	**37.40**	**53.95**	**66.85**
	Step-dLLM	**100.00**	99.60	37.20	95.60	**99.60**	33.60	52.30	60.95

Table 7: Per-sub-task RULER results on Dream-7B-Instruct (accuracy in %). Within each sparsity block, **bold** marks the highest score per sub-task. Step-dLLM rows are highlighted.

Sparsity	Method	VT	CWE	FWE	QS	QH
Dense		96.32	87.28	79.47	79.57	64.00
50%	StreamingLLM	13.40	43.34	37.73	**81.40**	33.00
	SparseD	96.28	**85.16**	**75.80**	80.37	**64.60**
	Step-dLLM	**96.40**	83.24	75.53	79.80	63.80
80%	StreamingLLM	0.36	4.54	4.33	19.18	19.60
	SparseD	95.44	71.76	**71.77**	**79.50**	61.00
	Step-dLLM	**96.04**	**73.24**	71.13	78.53	**61.80**
90%	StreamingLLM	0.00	0.00	0.00	0.00	0.00
	SparseD	92.20	54.72	67.53	78.45	**61.60**
	Step-dLLM	**94.68**	**59.58**	**68.26**	**79.03**	60.20
95%	SparseD	78.64	43.84	62.26	77.60	60.00
	Step-dLLM	**82.08**	**46.90**	**62.87**	**77.83**	**61.40**

B Additional Accuracy–Efficiency Results

We further evaluate the accuracy–latency trade-off. This sweep uses GSM8K, LLaDA-1.5, approximately 1K sequence length, and $f = 0.2$. Table 8 pairs accuracy with end-to-end inference time per denoising step. These are model-level ms/step measurements, not attention-kernel timings or full-generation durations. The 8K-context generation speedups in Figure 6 use a separate setup and should not be substituted into this table.

Table 8: Joint accuracy–latency sweep on GSM8K (LLaDA-1.5, approximately 1K sequence length). Accuracy is in percent; lower ms/step is better. Dense FlashAttention reaches 81.04% at 50.39 ms/step.

	Step-dLLM		SparseD	
Sparsity	Accuracy	ms/step	Accuracy	ms/step
50%	81.65	51.23	80.59	59.94
60%	82.34	50.21	74.68	59.53
70%	80.74	48.50	63.15	59.15
75%	78.92	48.19	62.02	58.53
80%	77.56	47.36	53.68	58.06
85%	74.91	47.57	42.76	57.69
90%	61.79	47.94	29.64	57.21
95%	18.80	46.18	9.86	56.50

Step-dLLM is both more accurate and faster than SparseD throughout this sweep. Near dense accuracy, Step-dLLM at 70% sparsity takes 48.50 ms/step with 80.74% accuracy, while SparseD at 50% takes 59.94 ms/step with 80.59% accuracy ($59.94/48.50 \simeq 1.24\times$). Relative to dense, the corresponding speedup is only $50.39/48.50 \simeq 1.04\times$. At 50%, Step-dLLM is slightly slower than dense. Sparse access must gather selected K/V blocks rather than stream a uniform dense layout; varying per-tile block counts also introduce load imbalance. Mask construction adds amortized overhead at anchor steps. With only a $2\times$ ideal reduction in attention FLOPs, these costs can offset savings, especially for short sequences.

Accuracy occasionally exceeds the dense baseline (e.g., 82.34% at 60% versus 81.04% dense). Pruning weak or irrelevant connections is one possible explanation, but these single reported evaluations do not distinguish that hypothesis from evaluation variability. Small accuracy differences relative to dense should be interpreted conservatively. The sweep establishes one practical operating point, rather than generality across backbones, tasks, and lengths.

C Periodic and Drift-Triggered Anchors

We compare the periodic schedule with a key-representation drift trigger on GSM8K at $\rho = 80\%$ and $S = 256$. At every denoising step, the trigger computes the average cosine distance between the current key vectors and the keys stored at the last anchor. If this distance exceeds τ, it performs full attention and rebuilds the mask, then updates the reference keys. Table 9 reports accuracy and the average number of mask refreshes per generation.

Table 9: Periodic versus key-drift-triggered anchors on GSM8K ($S = 256$, $\rho = 80\%$). Refresh count is an overhead indicator, not a measured latency ratio.

Configuration	Accuracy (%)	Avg. mask refreshes
Periodic $f = 0.2$	77.56	5
Adaptive key drift $\tau = 0.02$	82.03	24
Adaptive key drift $\tau = 0.001$	78.62	104.3

The threshold 0.02 improves accuracy by 4.47 percentage points, but requires $4.8\times$ as many refreshes. Lowering it to 0.001 increases refreshes further while reducing accuracy, so more frequent triggering alone is not a monotonic predictor of quality. Every trigger invokes a dense anchor, and measuring drift adds work at each step. These results motivate the periodic schedule's deterministic refresh

overhead; refresh counts quantify the number of dense anchors rather than a wall-clock latency ratio. Adaptive signals that are cheaper and better calibrated remain promising, especially for non-uniform decoding schedules. Alternating full attention and restricted reuse also appears in RefreshKV [28]; TeaCache [14] studies input-based cache decisions in video diffusion.

D Saliency Pooling Comparison

Table 10 compares max pooling with average pooling using the supplementary accuracy differences. Max pooling preserves a block whenever it contains a strong query–key connection, whereas averaging emphasizes aggregate affinity. Max pooling is favorable in the higher-sparsity measurements, with the largest reported difference at 95%; average pooling is favorable at 50% and 60%. Thus this design choice also depends on the operating regime. Quest [22] and SeerAttention [10] motivate examining extrema-based selection, while Native Sparse Attention [34] provides a complementary compression-based approach. This table contains the max-versus-average comparison only.

Table 10: Reported accuracy difference between max and average pooling, in percentage points. Positive favors max pooling. These are differences, not absolute accuracy values.

Sparsity	50%	60%	70%	80%	90%	95%
Max − average	-2.43	-0.46	$+0.76$	$+1.06$	$+0.83$	$+3.64$

E Composition with Fast-dLLM Dual Caching

Sparse attention reduces the query–key interactions evaluated within a forward pass. Cache-based decoding reduces repeated computation and can advance several output tokens per step. We combine Step-dLLM with Fast-dLLM's dual cache [26] on GSM8K with LLaDA and batch size 1. Table 11 reports the separately measured recomputation and cache-composition results. Its 80% recomputation accuracy (73.01%) belongs to this integration experiment and should not be substituted for the 77.56% in the main sweep.

Table 11: GSM8K accuracy (%) and decoding progress with dual caching. Relative tokens/step is the reported multiple of single-token progress, not wall-clock speedup. The 0% row uses the dense backbone.

Sparsity	Recompute ($f = 0.2$)	With dual cache	Relative tokens/step
0%	81.04	81.43	$3.80\times$
50%	81.65	80.06	$3.51\times$
80%	73.01	70.74	$2.71\times$

At 50% sparsity, dual caching reduces accuracy by 1.59 percentage points while reporting $3.51\times$ tokens per step. At 80%, the reduction is 2.27 points with $2.71\times$ tokens per step. This demonstrates compatibility in one integration setting, rather than a general wall-clock speedup or evidence that composition is lossless. Further evaluation with other cache methods and decoding schedules is left to future work.

F Models, Benchmarks, and Baseline Settings

Models. We evaluate pretrained LLaDA-1.5 [37] and Dream-7B-Instruct [30] without additional training. Step-dLLM changes the attention computation and refreshes the sparse pattern; non-anchor steps recompute their query, key, and value representations. The attention block size is $b = 16$ unless explicitly ablated; the semi-autoregressive generation block length is $B = 32$, which is a different parameter. The default refresh interval ratio is $f = 0.2$, with integer anchor spacing required in implementation.

Short-form benchmarks. GSM8K evaluates grade-school mathematical reasoning, MATH-500 evaluates more challenging mathematical problem solving, and HumanEval evaluates code generation. We report the accuracy values recorded for the corresponding evaluations.

Long-context benchmarks. NIAH evaluates retrieval with single and multiple needles, keys, queries, and values. RULER additionally includes variable tracking, word extraction, and question answering. Appendix A lists the per-sub-task results. Their arithmetic means give the main NIAH and RULER columns. Long-context task difficulty and backbone behavior differ, so a single averaged column can hide sub-task exceptions.

Dense and StreamingLLM. Dense attention provides the unpruned backbone reference. StreamingLLM [27] retains sink tokens plus a sliding context window; our setup assigns sink tokens 10% of the total generation length and derives the remaining window from the sparsity budget. This causal-decoder prior is a useful contrast but is not designed to track bidirectional attention throughout denoising.

SparseD. We use the same (b, B, ρ) as Step-dLLM and an initial full-attention fraction of 5%. SparseD constructs its static sparse mask after this warm-up and reuses it. Its block-sparse stage uses `flex_attention`, compiled into Triton kernels by TorchInductor. The refresh-interval ablation separately adds periodic refresh to this uniform-head-budget baseline; those rows must not be interpreted as the unchanged original SparseD algorithm.

Budgets and compute. All reported experiments use a minimum allocation of one key block per head and query block. With K the pre-floor top-K budget, the retained-block count is $K + \sum_h \max(0, 1 - c_h(p))$ at query block p; the minimum allocation therefore relaxes nominal sparsity whenever a head wins no entries. All experiments use a single NVIDIA H100 GPU.