LoSA: Locality Aware Sparse Attention in Diffusion Language Models

Anonymous Authors[1]

Abstract

Block-wise diffusion language models (DLMs) generate multiple tokens in parallel, offering a promising alternative to autoregressive decoding pipeline. However, they still remain bottlenecked by memory-bound attention in long context scenarios. Naïve sparse attention fails on DLMs due to the KV Inflation problem, where different queries select different prefix positions, making the union of accessed KV pages large. To solve this problem, we observe that block-wise diffusion exhibits locality of representation changes across denoising steps: only a small fraction of tokens' (active tokens) hidden states change significantly, while most tokens' (stable tokens) hidden states remain nearly constant. Based on this insight, we propose LoSA (**Lo**cality-aware **S**parse **A**ttention), which reuses cached prefix-attention results for stable tokens and applies sparse attention only to tokens with large representation changes. This reduces the number of queries that contributes to the union of KV indices and substantially shrinks the KV indices we need to load. Across multiple block-wise DLMs and reasoning benchmarks, LoSA maintains near-dense accuracy while improving efficiency, achieving up to $4.14\times$ speedup for the attention module over dense attention on RTX A6000 GPUs. LoSA also achieves 5% improvement than the baseline across all datasets and configurations, demonstrating the effectiveness of the purposed method.

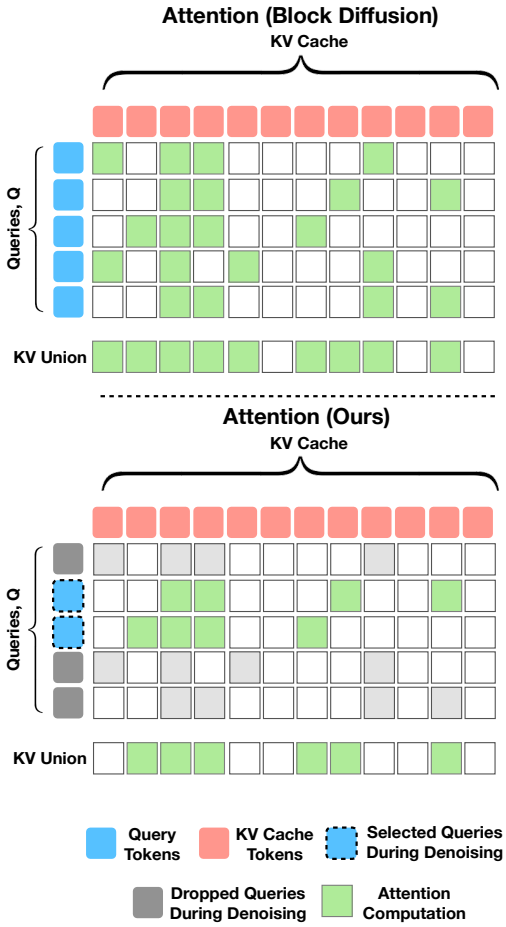

Figure 1. Illustration of the KV-union effect. In block diffusion, each query independently selects a small set of prefix KV positions, but the attention kernel must load the union of all selected positions across the block, leading to inflated KV access. Our method computes sparse attention only for selected (active) tokens during denoising and reuses cached results for the others (stable tokens), substantially reducing the size of the KV union and the latency.

1. Introduction

Diffusion language models (DLMs) have emerged as a compelling alternative to autoregressive transformers for text generation (Austin et al., 2021; Sahoo et al., 2024; Lou et al., 2024; Nie et al., 2025). Unlike autoregressive models that generate tokens sequentially, DLMs can produce multiple tokens in parallel through iterative denoising, offering potential speedups for short sequences and, combined with KV cache, could accelerate long sequence generations. Recent block-wise DLMs (Arriola et al., 2025; Cheng et al., 2025; Ye et al., 2025a) combine the strengths of both paradigms: they generate a *block* of tokens at each step via

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

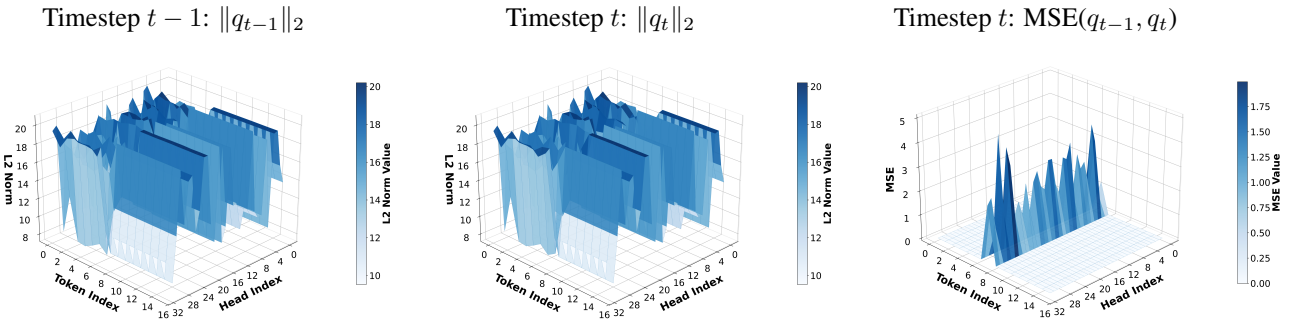

Figure 2. Visualizing representation change locality across denoising steps. Left and middle plots show the L2 norm of query vectors at timesteps $t-1$ and t respectively. The right plot shows the per-token MSE between query vectors across the two timesteps. Only a small fraction of tokens exhibit large changes, motivating the reuse of cached-prefix attention for stable tokens.

diffusion while maintaining autoregressive dependencies across blocks. Despite their parallelism advantages, block-wise DLMs face a critical efficiency bottleneck common to long-context models: attention to the cached prefix. At each denoising step, every token in the current block must attend to the entire key-value (KV) cache of previously generated tokens. For long sequences, loading this cache dominates inference latency since attention is memory-bound rather than compute-bound (Dao et al., 2022).

A natural approach is to apply sparse attention methods developed for autoregressive LLMs (Tang et al., 2024; Zhang et al., 2023; Liu et al., 2023; Xiao et al., 2024). These methods select a subset of KV positions per query, reducing memory traffic. However, when naively adapting such methods to diffusion language models (DLMs), we encounter a **KV Inflation problem**. In block-wise decoding, different queries within the same block tend to select different KV positions. Since the attention computation remains memory-bounded, the attention kernel must load the **union** of all required KV positions by any query in the block. As a result, when each of the B queries selects k KV positions, the size of this union can grow to $\min(L, Bk)$, where L is the prefix length, effectively negating the intended memory savings from sparsity.

To mitigate this problem, we observe that block-wise diffusion provides a unique structure absent in autoregressive decoding: **Locality of Representation Changes** across denoising steps. As shown in Figure 2, between consecutive denoising iterations, only a small fraction of tokens undergo substantial updates to their hidden representations; we refer to these as **Active tokens**. The remaining **Stable tokens** have queries that change minimally, meaning their attention patterns, and therefore their attention outputs, remain approximately constant. This locality structure enables us to reduce the number of queries participating in sparse KV selection, thereby mitigating the KV Inflation problem.

Building on this observation, we introduce LoSA (**Lo**cality-aware **S**parse **A**ttention), a method that exploits locality of representation changes to reduce KV cache memory operations. The key idea is simple: rather than recomputing sparse attention for all queries at every denoising step, we cache attention outputs from the previous step and reuse them for stable tokens. Only the active tokens require fresh attention computation. By reducing the number of queries that participate in sparse attention from B to $|\mathcal{S}|$ (where $\mathcal{S}$ is the set of active tokens), we shrink the union of required and directly lower memory traffic, which leads to higher speedup.

Beyond efficiency gains, LoSA also improves accuracy compared to naive sparse attention. The key advantage is that for stable tokens, our method preserves the full attention information from the previous step, whereas sparse attention can only access a subset of KV Cache. On the contrary, LoSA cache the information from the previous decoding steps, which leads to lower error. This reduction in error leads to higher accuracy while maintaining a low attention density.

We evaluate LoSA on the SDAR (Cheng et al., 2025) and Trado (Wang et al., 2025) model families. Our method achieves average of 5% accuracy improvement while maintaining $1.5\times$ lower attention density compared to baseline sparse attention method. On NVIDIA RTX A6000 GPUs, LoSA delivers a maximum of $4.14\times$ attention computation speedup, demonstrating its practical effectiveness for efficient block-wise DLM inference.

We summarize our contributions as follows:

- We observe **Locality of Representation Changes** in block-wise diffusion: across denoising steps, only a small fraction of tokens (**Active tokens**) undergo substantial representation changes, while the majority (**Stable tokens**) maintain approximately constant. Details in § 3.2.

- We identify the **KV Inflation problem** in block-wise DLM inference: when applying per-query sparse attention

to a block of B queries, the attention kernel must load the *union* of selected prefix KV positions across the block, so the effective KV load can grows up to $B\times$ and negate the intended speedup. Details in § 3.1.

- We propose LoSA, a sparse attention method that reuses cached prefix attention outputs for stable tokens and computes sparse attention only for active tokens, reducing the union of selected KV indices by $\sim 1.5\times$ in practice. Details in § 4.
- We demonstrate that LoSA achieves 5% higher accuracy than baseline sparse attention while delivering $4.14\times$ attention speedup over dense attention on NVIDIA RTX A6000 GPUs.

2. Preliminaries

2.1. Block-Wise Diffusion Language Models

Block-wise diffusion language models (DLMs) generate text in blocks of B tokens, where each block is treated as a single generation unit. Blocks are generated autoregressively, while attention within each block is bidirectional.

At a denoising step, the queries, keys and values corresponding to the current block are denoted as $\boldsymbol{Q}_b, \boldsymbol{K}_b, \boldsymbol{V}_b \in \mathbb{R}^{B\times d}$, where d is the head dimension and the subscript b stands for *block*. Since the denoising process follows an autoregressive manner, we maintain a KV cache of length L to avoid recomputation. We denote the cached prefix keys/values as $\boldsymbol{K}_p, \boldsymbol{V}_p \in \mathbb{R}^{L\times d}$, where the subscript p stands for *prefix*. Note that $\boldsymbol{K}_p$ and $\boldsymbol{V}_p$ are fixed across denoising steps, while $\boldsymbol{K}_b$ and $\boldsymbol{V}_b$ are updated at each step.

Consider a single attention head. The attention computation can be written as

$$\boldsymbol{O} = \mathrm{softmax}\left(\frac{\boldsymbol{Q}_b[\boldsymbol{K}_p, \boldsymbol{K}_b]^\top}{\sqrt{d}}\right)[\boldsymbol{V}_p, \boldsymbol{V}_b]. \tag{1}$$

2.2. Attention Decomposition With Online Softmax

Starting from the full attention computation, we decompose the computation into a prefix part and a suffix part using the online-softmax method adopted in FlashAttention (Dao et al., 2022). We track each part's log-normalizer (log-sum-exp) and attention output first, then merge them together. To simplify notation, we consider a single attention head and assume the query $\boldsymbol{q} \in \mathbb{R}^d$ has only one token.

We define the score operator $\mathcal{S}$ as:

$$\mathcal{S}(\boldsymbol{q}, \boldsymbol{K}) = \frac{\boldsymbol{q}\boldsymbol{K}^\top}{\sqrt{d}}.$$

Applying this operator to the prefix and current block yields score vectors $\boldsymbol{s}_p = \mathcal{S}(\boldsymbol{q}, \boldsymbol{K}_p) \in \mathbb{R}^L$ and $\boldsymbol{s}_b = \mathcal{S}(\boldsymbol{q}, \boldsymbol{K}_b) \in \mathbb{R}^B$. From these scores, we compute the corresponding log-normalizers L (which is a scalar):

$$L_p = \log \sum \exp(\boldsymbol{s}_p), \qquad L_b = \log \sum \exp(\boldsymbol{s}_b).$$

The attention outputs computation can also be rewritten using the log-normalizers:

$$\boldsymbol{o}_p = \mathrm{softmax}(\boldsymbol{s}_p)\,\boldsymbol{V}_p = \frac{1}{e^{L_p}} \exp(\boldsymbol{s}_p)\,\boldsymbol{V}_p \in \mathbb{R}^d, \tag{2}$$

$$\boldsymbol{o}_b = \mathrm{softmax}(\boldsymbol{s}_b)\,\boldsymbol{V}_b = \frac{1}{e^{L_b}} \exp(\boldsymbol{s}_b)\,\boldsymbol{V}_b \in \mathbb{R}^d. \tag{3}$$

We store the local log-normalizers L_p and L_b, together with the attention outputs $\boldsymbol{o}_p$ and $\boldsymbol{o}_b$, then use them using the online-softmax technique to obtain the final attention output.

When the query is a matrix $\boldsymbol{Q}_b \in \mathbb{R}^{B\times d}$, the size of the log-normalizers is $\mathbb{R}^B$ and the attention outputs is $\mathbb{R}^{B\times d}$, both are independent of L, meaning that the memory complexity is minimal when context length is long.

2.3. Sparse Attention Algorithms for LLMs

For LLMs, sparse attention methods approximate dense attention over the prefix by selecting a small subset of prefix positions. Given query $\boldsymbol{q}_i$, a sparse selector returns an index set $\mathcal{I} \subseteq \{1, \ldots, L\}$ with budget $|\mathcal{I}| = k$, and the attention output is computed only over the selected keys/values. In this paper, we use the QUEST algorithm and naively adapted it for DLMs as a baseline to compare against. QUEST partitions the KV cache into contiguous pages of size g and maintains lightweight min and max keys for each page. At inference time, it computes a query-dependent upper bound on the attention score for each page and selects the top $\frac{k}{g}$ pages for exact attention computation. Since the min and max keys are $g\times$ smaller than the original KV cache, the selection overhead is $g\times$ smaller than dense attention, which is minimal when the page size is large. Note that LoSA (introduced in § 4) is largely orthogonal to the choice of sparse selector algorithm.

3. Motivation

3.1. The KV Inflation Problem

In autoregressive LLM decoding, sparse attention algorithms will let one query selects k KV positions, and its attention computation latency is usually proportional to k, meaning that the reduction in the number of KV vectors loaded can effectively translate to speedup.

However, for DLMs, the situation is different. Suppose at each denoising step, the model processes a *block* of B query tokens. The attention workload consists of B query tokens attending to $L + B$ KV cache tokens. A naive adaptation of sparse attention applies a fixed budget k *per query token*

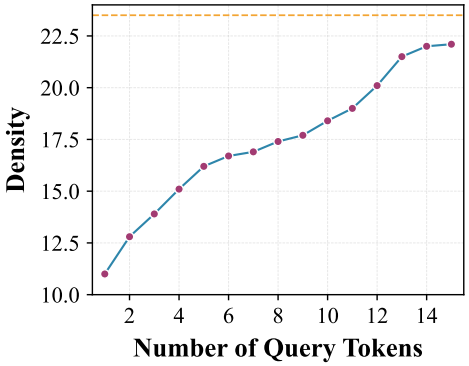

Figure 3. Illustration of KV-cache load reduction with locality-aware sparse prefix attention. We only need to load the prefix KV positions in the union $\mathcal{I} = \bigcup_{i \in \mathcal{S}} \mathcal{I}_i$ for the changed-token set $\mathcal{S}$, rather than the union over all B queries. This reduces the number of KV cache blocks/tiles fetched in memory-bound attention kernels.

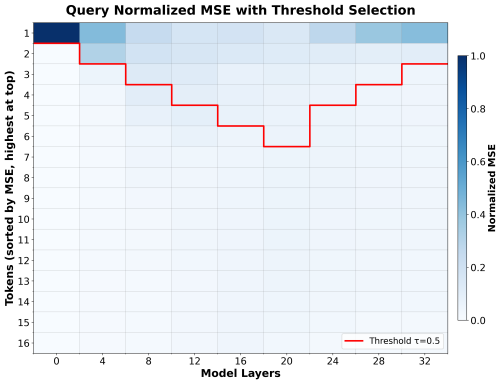

Figure 4. Visualizing locality across denoising steps. We plot the distribution of per-token MSE changes in queries between steps $t-1$ and t. For each layer, we sort the tokens in decreasing order of MSE changes from top to bottom in this heatmap. We observe that only a small fraction of tokens exhibit large changes (we show a threshold line which corresponds to 50% of the total MSE change is exhibited by tokens above that line), motivating the reuse of cached-prefix attention for the stable tokens.

in the block, producing index sets $\{\mathcal{I}_i\}_{i=1}^{B}$ with $|\mathcal{I}_i| = k$ for each query i. To ensure all important information is captured, we compute the union of selected KV positions across all queries. The effective number of KV positions being loaded is denoted as L':

$$L' = \Big| \bigcup_{i=1}^{B} \mathcal{I}_i \Big|. \tag{4}$$

Since the block size B is typically not large enough to make attention compute-bound, L' directly determines the memory traffic and thus the attention computation latency. The key insight is that loading a KV position incurs similar overhead whether it is used by one query or all queries in the block, making L' the critical factor for performance.

In this case, we identify the **KV Inflation problem**: this naive approach makes sparse attention extremely inefficient for DLM workloads. Even though each query selects only k KV positions, different queries may select different positions. Therefore, L' can become much larger than the intended budget k, negating the memory savings from sparsity. This phenomenon is well illustrated in Figure 1.

In the worst case, L' can grow to $\min(L, Bk)$, which can be much larger than the intended budget k. This large union size increases memory traffic and leads to sub-optimal speedup performance. As indicated in Figure 3, the KV Inflation problem can leads to $2.0\times$ higher attention density.

3.2. Locality of Representation Changes Across Denoising Steps

Our solution begins by observing that block diffusion decoding offers an additional structure that autoregressive LLM decoding does not: *locality of representation changes across denoising steps*. Between two consecutive denoising steps $t-1$ and t, only a small subset of tokens in the current block are updated from [MASK] to actual tokens. Tokens surrounding these updated positions experience significant changes in their hidden representations, while the remaining tokens' hidden states remain approximately constant.

Based on this observation, we categorize tokens into two groups: **active tokens**, which undergo substantial changes in their hidden representations, and **stable tokens**, which have approximately constant hidden states across consecutive denoising steps. We refer to this phenomenon as **Locality of Representation Changes**.

Quantifying locality for each token. To measure locality, we track the per-token change in queries, keys, and values using mean squared error (MSE). For $\boldsymbol{X} \in \{\boldsymbol{Q}, \boldsymbol{K}_b, \boldsymbol{V}_b\}$, let $\boldsymbol{x}^{(t)} \in \mathbb{R}^{B \times d}$ denote hidden states at denoising step t. We define the locality score as the per-token MSE between the current and previous step.

$$\Delta_x^{(t)} = \text{mean}\left(\frac{1}{d} \left\| \boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)} \right\|_2^2 \right)_{\text{token axis}} \in \mathbb{R}^B. \tag{5}$$

We visualize the distribution of $\Delta_x^{(t)}$ across tokens in Figure 4. The visualization confirms that active tokens exhibit large locality scores, while stable tokens have small scores. Notably, tokens at positions being decoded in the previous step show the largest changes.

We also observe that locality is layer-dependent. As shown in Figure 4, we sort the tokens by their locality scores and plot the distribution of locality scores for each layer. We observe that early layers and later layers tend to be more localized than middle layers.

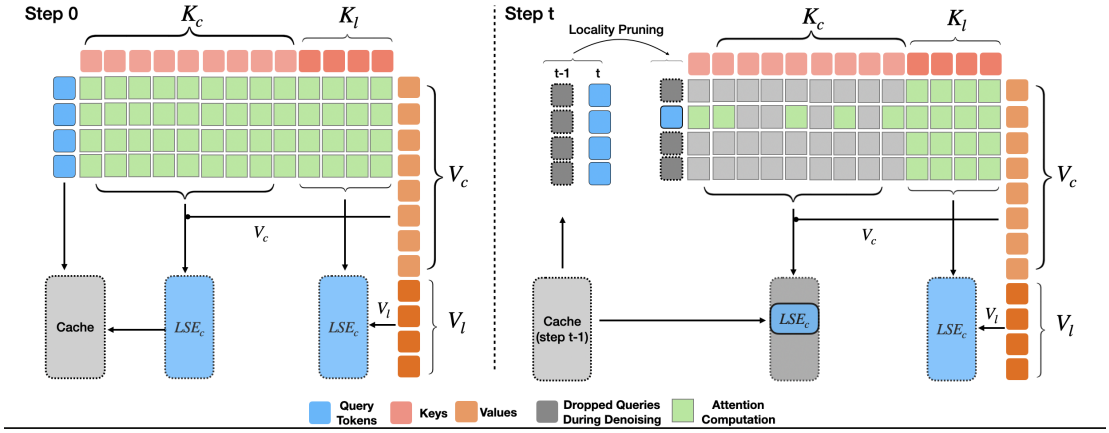

Figure 5. Overview of the locality-aware sparse prefix attention workflow. We (i) measure per-token query changes and select the changed-token set $\mathcal{S}$, (ii) run QUEST to obtain per-token prefix indices and take their union $\mathcal{I}$, (iii) load prefix KV for $\mathcal{I}$ and compute updated prefix statistics for queries in $\mathcal{S}$ while reusing cached statistics for stable tokens, (iv) compute dense within-block attention, and (v) merge prefix and block contributions via online softmax and cache the updated prefix statistics for the next denoising step.

4. Methodology

In this section, we describe the methodology of our LoSA method based on the aforementioned observations.

4.1. Locality Pruning

Let $\Delta \in \mathbb{R}^B$ denote the per-token locality score vector defined in equation 5. We rank tokens by their locality scores in descending order and select the top k tokens with the largest changes, forming the set $\mathcal{S}$ of **active tokens**. Tokens not in $\mathcal{S}$ are regarded as **stable tokens**.

4.2. Locality-Aware Sparse Attention for Prefix Tokens

We then propose a locality-aware sparse attention algorithm to reduce the KV cache loads. We cache and reuse the attention outputs for stable tokens, and only compute the attention for active tokens using sparse attention.

Reuse attention outputs for stable tokens. For stable tokens $i \notin \mathcal{S}$, since the query hidden states remain stable, the attention outputs for these tokens are also stable. Therefore, we can reuse the prefix-attention statistics $(\boldsymbol{o}_p, L_p)$ from the previous step:

$$(\boldsymbol{o}_p, L_p) \leftarrow (\boldsymbol{o}_p^{\text{prev}}, L_p^{\text{prev}}), \quad \text{token} \notin \mathcal{S}. \tag{6}$$

Sparse attention with smaller union size for active tokens. For active tokens $i \in \mathcal{S}$, we still apply the same sparse selector (QUEST) with budget k to select the prefix indices, but *only for the active tokens in* $\mathcal{S}$:

$$\mathcal{I}_i = \text{QUEST}(\boldsymbol{q}_i, \boldsymbol{K}_p; k), \quad i \in \mathcal{S}, \quad \mathcal{I} = \bigcup_{i \in \mathcal{S}} \mathcal{I}_i. \tag{7}$$

We load the prefix key/value vectors indexed by the union set $\mathcal{I}$. Then, for all queries in $\mathcal{S}$, we compute prefix attention over this shared set $\mathcal{I}$. After the computation, we update the prefix statistics $(\boldsymbol{o}_p, L_p)$ for these tokens:

$$(\boldsymbol{o}_p, L_p) \leftarrow (\boldsymbol{o}_p^{\text{current}}, L_p^{\text{current}}), \quad \text{token} \in \mathcal{S}. \tag{8}$$

4.3. Computation of the Suffix Tokens and Final Output

We next focus on the computation of $\boldsymbol{K}_b$ and $\boldsymbol{V}_b$. Although keys and values also exhibits the same locality structure as queries, their relevant attention computation is minimal since $B \ll L$. Therefore, we compute the attention between Q_b, K_b, and V_b densely, and get the low-dimensional statistics $(\boldsymbol{o}_b, L_b)$ for suffix tokens.

Finally, we merge the prefix and current block contributions using the online-softmax merge to obtain the final attention output. We also cache the updated prefix statistics $(\boldsymbol{o}_p, L_p)$ for the next denoising step. For the first denoising iteration for a block, cached statistics are unavailable; we fall back to dense prefix attention for initialization and starts to apply the locality-aware sparse attention from the second iteration onward. We summarize the overall workflow in Figure 5.

4.4. Why Locality Reduces KV Loads: Union Size

The KV-cache traffic for sparse prefix attention is governed by the number of unique prefix positions (or KV tiles) that must be fetched: $|\mathcal{I}| = \left|\bigcup_{i \in \mathcal{S}} \mathcal{I}_i\right|$. Compared to the naive per-token sparse scheme (which uses $\mathcal{S} = \{1, \ldots, B\}$), our locality-aware scheme uses a smaller set of queries, which can only reduce (never increase) the union:

$$\left|\bigcup_{i \in \mathcal{S}} \mathcal{I}_i\right| \leq \left|\bigcup_{i=1}^{B} \mathcal{I}_i\right| \leq \min(L, |\mathcal{S}|k). \tag{9}$$

KV Cache Per-Query Budget	Method	LongBench Accuracy HotPotQA	TriviaQA	NarrativeQA	Qasper	MultiFieldQA	Average (↑)
-	Dense	49.45%	84.79%	19.04%	17.75%	53.29%	44.86%
128	QUEST	29.17%	69.21%	7.46%	13.64%	38.21%	31.54%
	LoSA	48.27%	83.79%	17.19%	15.58%	45.00%	**41.97%** (+10.43)
256	QUEST	32.95%	75.23%	8.50%	14.04%	42.46%	34.64%
	LoSA	44.53%	81.82%	19.42%	17.11%	47.81%	**42.14%** (+7.50)
512	QUEST	34.58%	79.33%	8.99%	16.57%	44.04%	36.70%
	LoSA	44.19%	84.97%	18.39%	13.30%	50.14%	**42.20%** (+5.50)
1024	QUEST	39.88%	78.84%	7.37%	17.35%	45.00%	37.69%
	LoSA	48.45%	82.32%	18.89%	15.78%	48.94%	**42.88%** (+5.19)

Table 1. Accuracy (%) on LongBench under different retrieval budgets. LoSA consistently outperforms QUEST across all datasets and budget settings, with particularly large gains in low-budget regimes.

KV Cache Per-Query Budget	Method	LongBench Accuracy HotPotQA	TriviaQA	NarrativeQA	Qasper	MultiFieldQA	Average (↑)
128	QUEST	2.93%	3.80%	1.90%	6.72%	5.87%	4.24%
	LoSA	1.71%	2.29%	0.98%	4.15%	3.77%	2.58% (1.64×)
256	QUEST	5.84%	7.07%	3.89%	12.47%	11.14%	8.08%
	LoSA	3.37%	4.47%	2.05%	7.90%	7.43%	5.04% (1.60×)
512	QUEST	10.74%	13.06%	7.24%	23.13%	19.71%	14.78%
	LoSA	6.63%	8.54%	3.99%	15.39%	14.00%	9.71% (1.52×)
1024	QUEST	19.10%	22.85%	13.06%	39.06%	34.21%	25.66%
	LoSA	12.57%	16.24%	7.50%	28.81%	26.04%	18.23% (1.41×)

Table 2. KV-cache density (percentage of prefix tokens selected) on LongBench under different budgets. On all datasets and budget configurations, LoSA achieves an average of $1.54\times$ lower KV-cache density than the baseline.

The reduction is generally *not* proportional to $|\mathcal{S}|$: it depends on the overlap structure among the $\{\mathcal{I}_i\}$. Nevertheless, shrinking the participating query set from B to $|\mathcal{S}|$ eliminates many query-induced selections and typically decreases the number of KV cache that must be loaded. As shown in Figure 3, a smaller union size leads to smaller KV cache loads and higher speedup.

4.5. Efficiency Analysis

The overhead introduced by LoSA consists of three main components: (1) locality pruning to identify active tokens, (2) Deciding which tokens to compute attention for using the sparse selector and compute its union, and (3) computing the final attention output.

The locality pruning step requires calculating the per-token locality scores and ranking tokens by their scores, which incurs $\mathcal{O}(B \times d + B \log B)$ overhead. The sparse selector incurs $\mathcal{O}(\frac{B}{g} \times d)$ overhead, where g is the page size used by QUEST. Computing their union incurs $\mathcal{O}(\frac{B}{g})$ overhead. The final attention output computation also incurs $\mathcal{O}(B \times d)$ overhead, as the online-softmax merge is applied to LSE and local attention outputs. The total overhead is $\mathcal{O}(B \times d + \frac{B}{g} \times d)$. Since $B \ll L$ in typical DLM settings, these overheads are minimal compared to the actual attention computation.

Budget	Method	HellaSwag	WinoGrande	BoolQ	Avg.
–	Dense	69.60	63.60	72.92	68.71
128	QUEST	67.60	66.80	73.96	69.45
	LoSA	70.80	66.00	72.92	69.91
256	QUEST	73.60	63.20	75.00	70.60
	LoSA	70.40	65.20	73.96	69.85

Table 3. Accuracy (%) on three reasoning benchmarks. We report Dense, QUEST, and LoSA (excluding pure LoSA).

5. Experiments and Results

5.1. Experimental Settings

Models. We evaluate LoSA on two block-wise diffusion language models: Trado-8B-Instruct (Wang et al., 2025) and SDAR-8B-Instruct (Cheng et al., 2025). Both models follows the semi-autoregressive block-wise generation paradigm. We set the block size to 16 if not specified.

Baselines. Our primary baseline is QUEST (Tang et al., 2024), which we adapt for DLM workloads by applying sparse attention independently to each query in a block.

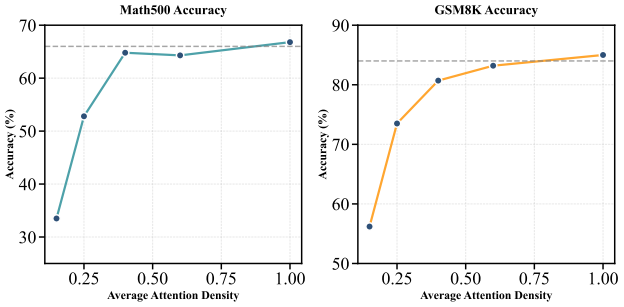

Figure 6. Accuracy results on mid-length reasoning benchmarks (GSM8k and MATH500) under different attention density settings. LoSA achieves competitive or better accuracy compared to QUEST across all budgets.

Datasets. We evaluate on mainly on LongBench (Bai et al., 2024) to examine their long-context ability. To be specific, we evaluate on HotPotQA (Yang et al., 2018), TriviaQA (Joshi et al., 2017), NarrativeQA (Kočiskỳ et al., 2018), Qasper (Dasigi et al., 2021), MultiFieldQA (Bai et al., 2024), and their average. We also evaluate on commonsense reasoning benchmarks, including HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), and BoolQ (Clark et al., 2019). We also evaluate on mathematical reasoning tasks, including GSM8K (Cobbe et al., 2021) and MATH500 (Hendrycks et al., 2021).

Implementation We implement our algorithm using customized CUDA and Triton (Tillet & Cox, 2019) kernels, together with attention kernels from FlashInfer (Ye et al., 2025b). We pick the top-5 query tokens in our locality pruning algorithm.

5.2. Results on Trado-8B-Instruct

5.2.1. LongBench Results

We first evaluate LoSA on LongBench. As shown in Table 1, LoSA significantly outperforms QUEST across all budget configurations and datasets. When the budget is 1024 and 512, LoSA achieves an average of 5% accuracy improvement than QUEST. When the budget is 256 and 128, LoSA accuracy gain is more significant, achieving 7.50% and 10.43% accuracy improvement respectively. Notably, LoSA maintains accuracy close to the Dense baseline even at aggressive sparsity levels, demonstrating that locality-aware reuse effectively preserves important attention information.

In terms of attention density, LoSA maintains an average of $1.54\times$ lower attention density than QUEST across all budget configurations and datasets. When the budget is 1024 and 512, LoSA attention density is $1.54\times$ lower than QUEST. When the budget is 256 and 128, LoSA attention density is $1.54\times$ lower than QUEST.

5.2.2. Commonsense Reasoning Results

We further evaluate LoSA on commonsense reasoning benchmarks. Table 3 shows accuracy results under different budgets. LoSA achieves competitive or better accuracy compared to QUEST across all three datasets. At budget 128, LoSA achieves an average accuracy of 69.91%, slightly outperforming QUEST's 69.45%. At budget 256, while QUEST achieves 70.60% average accuracy, LoSA maintains 69.85%, demonstrating consistent performance across different sparsity levels. Figure 6 visualizes these results, showing that LoSA maintains accuracy comparable to QUEST while benefiting from reduced KV cache density.

5.3. Results on SDAR-8B-Instruct

We validate LoSA on SDAR-8B-Instruct using GSM8K and MATH500, two mathematical reasoning benchmarks. To understand the accuracy-density trade-off, we evaluate multiple threshold settings that control the sparsity level. As the threshold increases, attention density decreases, allowing us to observe how accuracy evolves with sparsity. LoSA demonstrates high accuracy retention even at significantly reduced attention densities, maintaining competitive performance with the Dense baseline while achieving substantial memory savings. The locality-aware approach enables LoSA to preserve critical attention patterns for stable tokens, leading to better accuracy-density trade-offs compared to naive sparse attention methods.

5.4. Latency Analysis

We measure end-to-end latency of prefix attention computation to validate that our method translates sparsity into wall-clock speedup. We evaluate two configurations: (1) 64K context length with 16 token block size, and (2) 32K context length with 32 token block size, both using Trado-8B-Instruct and testing on TriviaQA.

Figure 7 compares the latency breakdown of LoSA, QUEST, and Dense Attention. The latency breakdown consists of several components: (1) **Criticality Estimation**, which refers to the sparse attention selector (QUEST) that identifies relevant KV positions for each query; (2) **Locality Pruning**, which identifies active tokens by computing locality scores based on representation changes; (3) **Cache Management**, which stores and retrieves cached attention outputs for stable tokens; and (4) **Attention Computation**, which performs the actual sparse attention operations over selected KV positions.

LoSA achieves a $4.14\times$ speedup over Dense Attention, outperforming QUEST's $3.26\times$ speedup. The key advantage of LoSA is that locality-aware reuse substantially reduces the number of queries requiring fresh attention computation. By caching and reusing attention outputs for stable tokens,

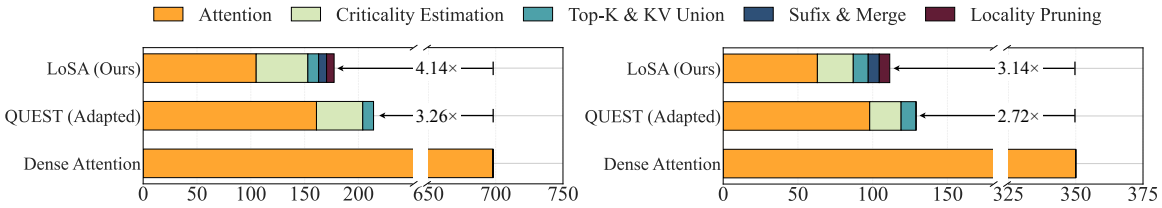

Figure 7. Latency breakdown comparison on prefix attention. LoSA achieves $4.14\times$ speedup over Dense Attention by reducing memory-bound attention operations through locality-aware sparsity. QUEST (Adapted) is our implementation of a representative sparse attention baseline for dLLMs, achieves $3.26\times$ speedup.

LoSA minimizes memory-bound operations while maintaining accuracy, leading to lower attention computation time despite the additional overhead of locality pruning and cache management.

6. Related Work

6.1. Diffusion Language Models

Diffusion models have achieved remarkable success in continuous domains such as images and audio, inspiring efforts to adapt them for discrete text generation. Early work explored discrete diffusion through absorbing states (Austin et al., 2021) and multinomial diffusion processes. More recently, masked diffusion language models (MDLM) (Sahoo et al., 2024) demonstrated that simple masked diffusion with modern architectures can match autoregressive baselines on language modeling benchmarks. SEDD (Lou et al., 2024) introduced score entropy-based training for discrete diffusion, achieving state-of-the-art perplexity. LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025a) scaled masked diffusion models demonstrating competitive performance with autoregressive models of similar scale.

Block-wise diffusion models. Recent work has explored semi-autoregressive diffusion that generates tokens in blocks. BD3-LMs (Arriola et al., 2025) interpolate between fully autoregressive and fully parallel generation by producing fixed-size blocks autoregressively while using diffusion within each block. SDAR (Cheng et al., 2025) converts an autoregressive LLM into a block-diffusion language model. Trado (Wang et al., 2025) applies advanced post-training techniques to enhance their reasoning capabilities. Our work is designed for this block-wise setting.

6.2. Sparse Attention for Long-Context LLMs

Reducing the quadratic complexity of attention has been extensively studied. Fixed sparse patterns such as local windows combined with global tokens (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020) reduce complexity to linear but require architectural modifications during training. Content-based sparse attention methods like Reformer (Kitaev et al., 2020) use locality-sensitive hashing to identify relevant keys. Linear attention variants (Katharopoulos et al., 2020; Choromanski et al., 2021) approximate softmax attention with kernel feature maps, enabling recurrent computation.

For pretrained LLMs at inference time, dynamic sparse attention methods select relevant KV positions per query without retraining. QUEST (Tang et al., 2024) groups KV cache into pages and selects pages based on estimated attention scores. H_2O (Zhang et al., 2023) maintains a dynamic cache of "heavy hitter" tokens that accumulate high attention mass. Scissorhands (Liu et al., 2023) exploits the persistence of important tokens across generation steps. StreamingLLM (Xiao et al., 2024) discovers that keeping initial "sink" tokens enables stable streaming inference. SnapKV (Li et al., 2024) and PyramidKV (Cai et al., 2024) compress the KV cache based on attention patterns observed during prefill. MInference (Jiang et al., 2024) accelerates long-context prefilling through dynamic sparse patterns.

6.3. KV Cache Optimization and Efficient Serving

Beyond sparse attention, several orthogonal techniques improve KV cache efficiency. FlashAttention (Dao et al., 2022; Dao, 2024) optimizes memory access patterns through tiling, reducing memory traffic for dense attention without approximation. PagedAttention (Kwon et al., 2023) enables non-contiguous KV cache storage, improving memory utilization in serving systems.

7. Conclusion

In this paper, we identify the KV Inflation problem in block-wise diffusion language models as the main reason why naive sparse attention fails to accelerate block-wise DLMs. We observe locality of representation changes across denoising steps and propose LoSA, which reuses cached attention for stable tokens and computes sparse attention only for active tokens. This reduces the number of queries participating in sparse KV selection, effectively mitigating the KV Inflation problem. LoSA achieves 5%–10.43% accuracy improvements over QUEST on LongBench with $1.54\times$ lower attention density, and a $4.14\times$ speedup on A6000 GPUs.

Impact Statement

This paper presents work that aims to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Arriola, M., Gokaslan, A., Chiu, J. T., Yang, Z., Qi, Z., Han, J., Sahoo, S. S., and Kuleshov, V. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. 1, 8

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and van den Berg, R. Structured denoising diffusion models in discrete state-spaces. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 17981–17993, 2021. 1, 8

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024. 7

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *ArXiv preprint*, abs/2004.05150, 2020. 8

Cai, Z., Zhang, Y., Gao, B., Liu, Y., Liu, T., Lu, K., Xiong, W., Dong, Y., Chang, B., Hu, J., and Xiao, W. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *ArXiv preprint*, abs/2406.02069, 2024. 8

Cheng, S., Bian, Y., Liu, D., Zhang, L., Yao, Q., Tian, Z., Wang, W., Guo, Q., Chen, K., Qi, B., et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*, 2025. 1, 2, 6, 8

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. 8

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. 8

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019. 7

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 7

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. 8

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. 2, 3, 8

Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021. 7

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 7

Jiang, H., Li, Y., Zhang, C., Wu, Q., Luo, X., Ahn, S., Han, Z., Abdi, A., Li, D., Lin, C., Yang, Y., and Qiu, L. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. 8

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017. 7

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020. 8

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 8

Kočiskỳ, T., Schwarz, J., Blunsom, P., Dyer, C., Hermann, K. M., Melis, G., and Grefenstette, E. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. 7

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180. 8

Li, Y., Huang, Y., Yang, B., Venkitesh, B., Locatelli, A., Ye, H., Cai, T., Lewis, P., and Chen, D. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024. 8

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. 2, 8

Lou, A., Meng, C., and Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. 1, 8

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., Zhou, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models. *ArXiv preprint*, abs/2502.09992, 2025. 1, 8

Sahoo, S. S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E., Chiu, J. T., Rush, A., and Kuleshov, V. Simple and effective masked diffusion language models. In Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J. M., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. 1, 8

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021. 7

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024. 2, 6, 8

Tillet, P. and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019. 7

Wang, Y., Yang, L., Li, B., Tian, Y., Shen, K., and Wang, M. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*, 2025. 2, 6, 8

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. 2, 8

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018. 7

Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., and Kong, L. Dream 7b: Diffusion large language models. *ArXiv preprint*, abs/2508.15487, 2025a. 1, 8

Ye, Z., Chen, L., Lai, R., Lin, W., Zhang, Y., Wang, S., Chen, T., Kasikci, B., Grover, V., Krishnamurthy, A., et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025b. 7

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. Big bird: Transformers for longer sequences. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 8

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019. 7

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023. 2, 8