NOSA: Native and Offloadable Sparse Attention

Yuxiang Huang [*1] **Pengjie Wang** [*1] **Jicheng Han** [1] **Weilin Zhao** [1] **Zhou Su** [2] **Ao Sun** [3] **Hongya Lyu** [2]
Hengyu Zhao [2 4] **Yudong Wang** [1 2] **Chaojun Xiao** [1] **Xu Han** [1] **Zhiyuan Liu** [1]

Abstract

Decoding throughput is often limited by GPU memory dominated by the KV cache. Existing KV cache offloading reduces memory by storing context on CPU and fetching sparse KV subsets, but training-free methods suffer from long-generation quality degradation, while trainable sparse attention incurs excessive CPU–GPU transfers. We propose NOSA, a trainable sparse attention mechanism natively designed for KV cache offloading. NOSA constrains CPU–GPU KV transfer volume to lower communication overhead and improve throughput. We further build NOSI, an offloading inference system that realizes NOSA's efficiency. Experiments on $\{1, 3, 8\}$B LLMs show that NOSA improves quality across general, long-input, and long-generation tasks, while boosting decoding throughput by up to $5.04\times$, $1.92\times$, and $1.83\times$ over FullAttn, InfLLMv2, and ShadowKV.

1. Introduction

Decoding efficiency of large language models (LLMs) (OpenAI, 2025; Anthropic, 2025; DeepSeek-AI, 2025a) is critical for advancing AI applications that require processing long-context inputs or outputs, such as LLM-based agents (Qin et al., 2024; Luo et al., 2025) and solving complex reasoning tasks (DeepSeek-AI, 2025b; Guo et al., 2025). However, this efficiency is heavily constrained by the $O(N)$-scale memory I/O overhead as the input length increases.

To improve decoding efficiency, batched inference with KV-cache offloading is widely adopted (Xiao et al., 2024a; Sun et al., 2025; Yang et al., 2025). These methods keep most key-value (KV) cache units on the CPU and fetch only a small subset to the GPU for attention, reducing GPU memory footprint and enabling larger batches and higher decoding throughput. However, their training-free subset selection introduces a mismatch between training and inference, as the sparse patterns have not been seen during training, leading to substantial performance degradation (Figure 1), particularly in long-generation scenarios where selection errors accumulate.

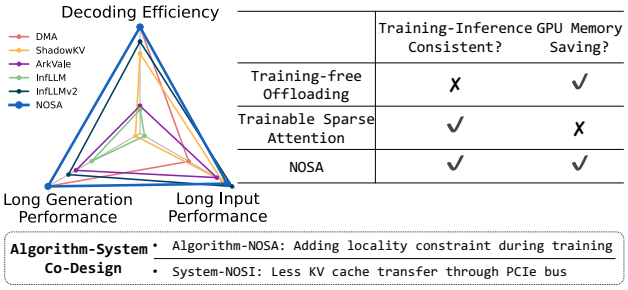


Figure 1. Overview of NOSA and NOSI.

Meanwhile, trainable sparse attention methods (Yuan et al., 2025; Lu et al., 2025; Zhao et al., 2025) can alleviate the training-inference gap, achieving near-lossless long-generation performance and outperforming prior attention-centric optimizations, including training-free sparse attention (Li et al., 2025; Xu et al., 2025) and KV-cache compression methods (Zhang et al., 2023b; Li et al., 2024). However, these trainable methods do not reduce the size of the KV cache since any KV entry may be attended by a query, and thus cannot increase the maximum batch size under GPU memory constraints. Therefore, ***designing a trainable sparse attention mechanism that is natively offloadable*** is essential to address these challenges.

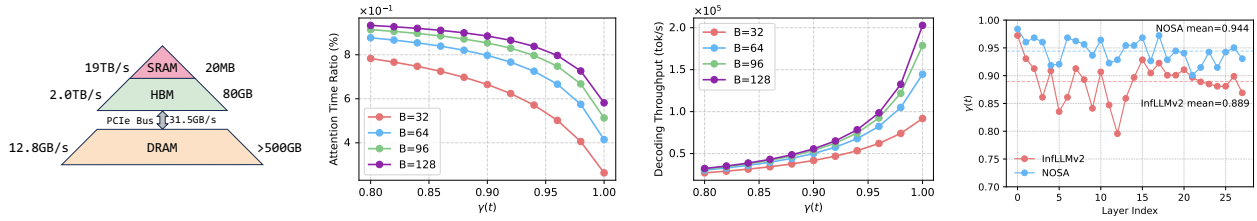
To this end, we propose NOSA and NOSI. We summarize our contributions as follows.

Contribution 1: We analyze the feasibility of KV cache offloading with trainable sparse attention. Using InfLLMv2 (Zhao et al., 2025) as a representative method, we identify strong block-selection locality and characterize the resulting PCIe bottleneck. Our analysis shows that such locality naturally enables KV reuse, but PCIe bandwidth still makes decoding communication-bound.

Contribution 2: Based on these insights, we introduce NOSA, a trainable sparse attention mechanism natively

[1]Tsinghua University [2]OpenBMB [3]Beijing University of Posts and Telecommunications [4]Beijing Institute of Technology. Correspondence to: Chaojun Xiao, Xu Han, Zhiyuan Liu <{xcj,hanxu,liuzy}@tsinghua.edu.cn>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).



(a) Memory Hierarchy *(b)* Attention Time Ratio *(c)* Theoretical Throughput *(d)* Locality $\gamma(t)$

Figure 2. The memory hierarchy of NVIDIA A800-80GB, the ratio of attention mechanism among the complete inference time, the theoretical decoding throughput with respect to various cache hit rates, and INFLLMV2's locality. "B" represents the batch size.

designed for offloading by explicitly limiting KV cache transmission. NOSA decomposes KV selection into query-aware and query-agnostic parts, and applies eviction over query-agnostic selection to bound CPU-fetched KV blocks. Experiments show that NOSA preserves task performance and consistently outperforms KV offloading baselines, especially on long-generation tasks.

Contribution 3: We further develop NOSI, an offloading system tailored for NOSA, since vanilla implementations fail to fully expose PCIe bottlenecks. With system-level optimizations, NOSI achieves higher decoding efficiency across settings, improving throughput by up to 5.04×, 1.92×, and 1.83× over FULLATTN, INFLLMV2, and SHADOWKV.

2. Background

2.1. Preliminaries

Trainable Sparse Attention. Transformer-based LLMs employ attention modules to capture cross-token relationships and long-range dependencies. The attention mechanism takes $\mathbf{H} \in \mathbb{R}^{n\times d}$ as input, where n and d denote the input length and model dimension, respectively. The query, key, and value matrices are then obtained through linear projections: $\mathbf{Q} = \mathbf{H}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{H}\mathbf{W}_K$, $\mathbf{V} = \mathbf{H}\mathbf{W}_V$. The attention computation is then formulated as below, where $\mathbf{M} \in \{0, -\infty\}^{n\times n}$ is the attention mask.

$$\mathbf{P} = \mathbf{M} + \mathbf{Q}\mathbf{K}^\top,\ \mathbf{A} = \mathrm{Softmax}(\mathbf{P}),\ \mathbf{O} = \mathbf{A}\mathbf{V}. \quad (1)$$

Trainable sparse attention methods commonly adopt a block-sparse attention pattern. Here, we briefly review the notations of INFLLMV2 (Zhao et al., 2025). Before performing the attention computation, the key matrix is first partitioned into blocks of size n_b, and each block is compressed into a single representation using a compression function $f_c : \mathbb{R}^{n_b\times d} \to \mathbb{R}^{1\times d}$. This process can be formalized as follows: $\mathbf{K}_c = f_c(\mathbf{K}) \in \mathbb{R}^{\frac{n}{n_b}\times d}$. The selection score $\mathbf{S}_c^q = (s_{ij}^q)_{n\times\frac{n}{n_b}}$ between each token and all blocks is then computed as: $\mathbf{S}_c^q = \mathbf{Q}\mathbf{K}_c^\top \in \mathbb{R}^{n\times\frac{n}{n_b}}$. Since INFLLMV2 also incorporates attention sinks and sliding windows, we denote their lengths as n_s and n_w, respectively. Then, the attention mask $\mathbf{M} = (m_{ij})_{n\times n}$ is computed as:

$$m_{ij} = 0, \mathbb{I}_{\text{cau}} \wedge (\mathbb{I}_{\text{sink}} \vee \mathbb{I}_{\text{win}} \vee \mathbb{I}_{\text{topk}});\ m_{ij} = -\infty, \text{others}; \quad (2)$$

where $\mathbb{I}_{\text{cau}} = i \geq j$; $\mathbb{I}_{\text{sink}} = j < n_s$; $\mathbb{I}_{\text{win}} = i - j < n_w$; $\mathbb{I}_{\text{topk}} = s_{ij}^q \in \mathrm{Top}_k\left(s_{i,\leq\lfloor i/n_b\rfloor}^q\right)$.

KV Cache Selection. KV cache selection is widely used in sparse attention mechanisms, where a limited number of KV entries are accessed and computed. Mainstream approaches to KV cache selection can be broadly categorized into two types: *query-aware selection* (Yuan et al., 2025; Zhao et al., 2025; Li et al., 2024) and *query-agnostic selection* (Kim et al., 2024; Huang et al., 2025; Yao et al., 2024).

For query-aware selection, we first compute the selection score $\mathbf{S}^q = \mathbf{Q}\mathbf{K}^\top$, and then choose the top-k positions, where $m_{ij} = 0$ if $s_{ij}^q \in \mathrm{Top}_k(s_{i,\leq i}^q)$. This utilizes information from the query to ensure the ability to recall distant information, but does not guarantee selection locality.

Query-agnostic selection is widely adopted in scenarios where the query tokens are not immediately available. To perform this selection, we first compute the importance score for each KV cache unit using a scoring function $s_i^e = f_e(\mathbf{h}_i) \in \mathbb{R}$, and then select the top-k positions, where $m_{ij} = 0$ if $s_j^e \in \mathrm{Top}_k(s_{\leq i}^e)$. Query-agnostic selection does not guarantee the ability to recall and yields a fixed eviction pattern, which maintains strong locality and enables zero communication when applied to offloading systems.

2.2. Observations

We conduct an analysis on applying trainable sparse attention to KV offloading on typical compute architectures, as shown in Figure 2a. We summarize two key observations.

Observation 1: Locality in Query-aware Selection

Trainable sparse attention exhibits inherent locality in query-aware token/block selection across consecutive decoding steps.

We analyze KV selection locality in INFLLMV2 (Zhao et al., 2025). Since KV cache offloading is mainly limited by PCIe transfer overhead, higher locality enables reusing KV entries selected in the previous decoding step, thereby reducing communication.

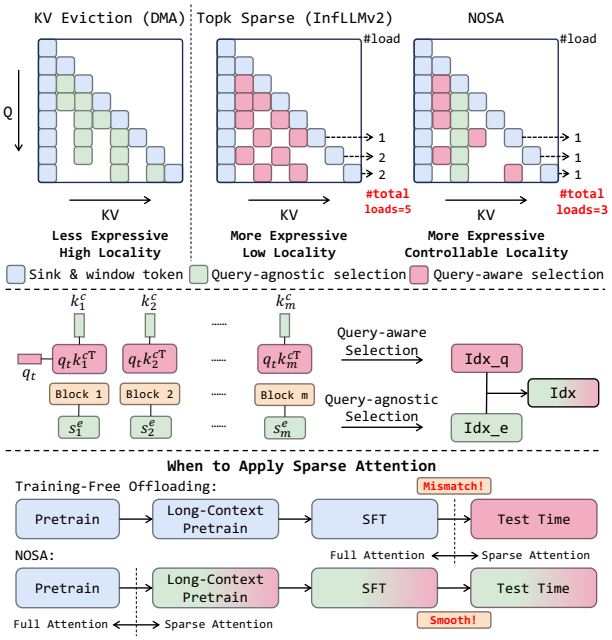


Figure 3. The framework of NOSA. By introducing a locality constraint, NOSA reduces the number of blocks loaded at each step. We apply NOSA from the long-context pretraining stage.

Definition 2.1 (Locality)**.** Given an attention mask $\mathbf{M} = (m_{ij})_{n\times n}$, the selected tokens at step t are $\Gamma(t) = \{i : m_{it} = 0\}$. Locality is defined as $\gamma(t) = \frac{|\Gamma(t)\cap\Gamma(t-1)|}{|\Gamma(t)|}$.

Using a 16K-token input with 4K selected tokens, Figure 2d shows that $\gamma(t) \geq 0.8$ for most layers. This indicates that fewer than 20% of KV entries change between adjacent steps, suggesting that trainable sparse attention naturally provides reuse opportunities for KV offloading.

Observation 2: PCIe Communication Bound

Query-aware offloading inference remains bounded by PCIe communication latency.

Despite the locality in Obs. 1, we find that $\sim 80\%$ cache reuse is still insufficient to hide PCIe overhead. As shown in Figure 2b, over 80% of decoding time is spent in attention, indicating that throughput remains communication-bound.

We further simulate theoretical latency and throughput under different cache hit rates following Yuan et al. (2024). Figure 2c shows that increasing cache hit rate substantially reduces attention time and improves decoding throughput.

Takeaway: Based on Obs. 1-2, high-throughput decoding requires a constraint that explicitly promotes higher cache hit rates, enabling natively offloadable sparse attention.

3. NOSA and NOSI

In this section, we first describe the algorithmic design of NOSA, followed by detailed ablations on core designs. The overall framework of NOSA is illustrated in Figure 3.

3.1. Offloading-aware Trainable Sparse Attention

Adding Locality Constraint. As discussed in Section 2.2, improving locality in token (or block) selection is the key to achieving an efficient offloading system. Accordingly, the core idea of NOSA is to improve selection locality during both training and inference. A natural approach is to enforce a minimum level of KV selection locality, formalized as follows. Aiming to maintain a desired locality $\gamma(t)$ across all token positions t, we impose a lower bound $\gamma_0 \in (0, 1)$ on the overlapping rate of selected tokens between adjacent decoding steps, i.e., we design an algorithm such that $\forall t \in \{2, \cdots, n\}, \gamma(t) \geq \gamma_0$.

To achieve this goal, we divide the selected tokens $\Gamma(t)$ into two subsets: the query-aware selection $\Gamma_q(t)$ and the query-agnostic selection $\Gamma_e(t)$. For query-aware selection $\Gamma_q(t)$, we do not add any constraints to preserve the model's performance. As described in Section 2.1, query-agnostic selection naturally imposes an eviction constraint, i.e., if a token is not selected at decoding step t, it must not be selected at any subsequent step $t' > t$, and $\Gamma_e(t') \subseteq \Gamma_e(t) \cup \{t+1, \cdots, t'\}$, which emerges naturally rather than being explicitly designed.

We implement the query-agnostic selection by using an *eviction head* to assign each KV pair an *importance score* $s_t^e \in \mathbb{R}$. When selecting k query-agnostic tokens, a Top-k function is applied to these scores, formulated as $\Gamma_e(t) = \mathrm{ArgTop}_k(\{s_i^e\}_{i=0}^t)$. To further enhance performance, we implement the eviction head using DMA (Shi et al., 2025), a trainable sparse attention mechanism with query-agnostic selection, which can be viewed as a trainable KV eviction method. The importance score is computed as $s_j^e = \tau(\mathbf{v}_j\mathbf{W}_1)\mathbf{W}_2$, where τ is a non-linear function, $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{kv}}\times n_{\text{kv}}}$, $\mathbf{W}_2 \in \mathbb{R}^{n_{\text{kv}}\times 1}$ are trainable parameters, and $d_{\text{kv}}, n_{\text{kv}}$ are the dimension and the number of key/value heads, correspondingly. τ is implemented as the `softplus` function in DMA's design. We remove the final exp operation from the original design to improve performance, as validated by the ablation studies in Appendix E.5. For the query-aware selection, we adopt $s_{tj}^q = \mathbf{q}_t\mathbf{k}_j^\top$ as the selection score to preserve the model's ability to retrieve previously evicted tokens, and also apply a Top-k function to these scores for selection.

Adapting to Block-wise Selection. Mainstream trainable sparse attention mechanisms can be broadly categorized into element-wise (DeepSeek-AI, 2025a) and block-wise approaches (Yuan et al., 2025; Zhao et al., 2025). The computation of s_j^e and s_{tj}^q described above follows an element-wise paradigm, where selection is performed at the token level. However, compared to block-wise patterns, element-wise selection is fundamentally inefficient for KV cache

offloading due to low PCIe bandwidth utilization, as shown in the ablation studies (Appendix E.5). Therefore, we adopt a two-stage, block-wise selection strategy and build NOSA on top of INFLLMV2 (Zhao et al., 2025). Specifically, mean pooling is first applied to sub-blocks within each block, followed by max pooling at the block level.

Denote the j-th block as $[j]$, and each block contains n_b tokens. We first perform mean pooling with a small stride l_s^{mean} and a kernel size l_k^{mean} on each sub-block. We denote the j-th sub-block as $\langle j \rangle$, and proceed as follows:

$$s^q_{i\langle j\rangle} = \frac{1}{l_k^{\text{mean}}} \sum_{t=j\times l_s^{\text{mean}}}^{j\times l_s^{\text{mean}}+l_k^{\text{mean}}} s^q_{it},\ s^e_{\langle j\rangle} = \frac{1}{l_k^{\text{mean}}} \sum_{t=j\times l_s^{\text{mean}}}^{j\times l_s^{\text{mean}}+l_k^{\text{mean}}} s^e_t \tag{3}$$

Then, a max pooling is applied to each block:

$$s^q_{i[j]} = \max_{\langle t\rangle\subset[j]} s^q_{i\langle t\rangle},\ s^e_{[j]} = \max_{\langle t\rangle\subset[j]} s^e_{\langle t\rangle} \tag{4}$$

At each decoding step, k/n_b blocks are selected. We assign two budgets k_q, k_e for query-aware and query-agnostic selection, such that $k = k_q + k_e$. We first select the query-aware blocks by $s^q_{i[j]}$. We then set these places to infinity and select the query-agnostic blocks by $s^e_{[j]}$. Such process follows taking a Top-k selection ($\Gamma(i) = \text{ArgTop}_{k/n_b}(\{s_{i[j]}\}_{j=1}^{\lfloor t/n_b\rfloor})$) on the following score:

$$s_{i[j]} = \begin{cases} +\infty, & s^q_{[j]} \in \text{Top}_{k_q/n_b}\left(\{s^q_{[j]}\}_{j=1}^{\lfloor i/n_b\rfloor}\right); \\ s^e_{[j]}, & \text{otherwise.} \end{cases} \tag{5}$$

This design inherently guarantees a lower bound on locality, as formalized in Theorem 3.1 (Proved in Appendix F).

Theorem 3.1. *If the above selection process has budget $k = k_q + k_e$, we have $\forall t \in \{2, \cdots, n\},\ \gamma(t) \geq \frac{k_e}{k}$.*

Attention Calculation. At the decoding step for generating the $(t+1)$-th token, the hidden input is $\mathbf{h}_t \in \mathbb{R}^{1\times d}$, with its corresponding query, key, and value vectors $\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t \in \mathbb{R}^{1\times d}$, and the KV cache $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{t\times d}$. We first compute the importance scores $\mathbf{S}^e = (s^e_j)_t$, then we utilize the selection process defined in Equation 5 to generate the selected positions $\Gamma(t)$ and the corresponding attention mask $\mathbf{M}$, where $m_j = 0$ if $j \in \Gamma(t)$ and $m_j = -\infty$ otherwise. To make this selection differentiable and trainable, we introduce an attention bias vector $\mathbf{b} \in \mathbb{R}^t$ similar to Shi et al. (2025), where each element is assigned as $b_j = s^e_j = \tau(\mathbf{v}_j\mathbf{W}_1)\mathbf{W}_2$.

Finally, the attention output $\mathbf{o}_t \in \mathbb{R}^d$ is computed as

$$\mathbf{o}_t = \sum_{j=1}^{t} \frac{\exp(b_j)\exp(\mathbf{q}_t\mathbf{k}_j^\top + m_j)}{\sum_{i=1}^{t}\left(\exp(b_i)\exp(\mathbf{q}_t\mathbf{k}_i^\top + m_i)\right)}\mathbf{v}_j. \tag{6}$$

We observe that the query-agnostic selection is highly sensitive to numerical precision. Therefore, we omit the exponential ($\exp$) operation before token selection; that is, we select the top-k indices based on pre-exponential importance score $s^e_{[j]}$ rather than on $\exp(b_j)$. Since this operation is deferred to the attention computation, we refer to this optimization as ED-DMA (Exp-Delayed DMA). As shown in the ablation studies in Appendix E.5, ED-DMA yields the most stable and effective results, whereas alternative approaches lead to noticeably greater performance degradation.

Training with NOSA. The design of NOSA is fully trainable, as the eviction head receives gradients through the attention bias $\mathbf{b}$. To equip LLMs with native offloading capability, *we incorporate NOSA throughout long-context adaptation*. Specifically, we pretrain with vanilla attention on short-context data, then switch to NOSA for long-context continual pretraining. We also apply NOSA during SFT, therefore delivering the model with built-in NOSA support.

3.2. NOSI: Inference System for NOSA

The vanilla implementation of NOSA based on Hugging Face Transformers (Wolf et al., 2020) is highly inefficient due to two factors due to inefficient kernels and excessive small kernel launch overheads. To mitigate these issues, we implement NOSI, a NOSA-based System for Inference, which exposes true communication overhead from other performance bottlenecks. We provide a pseudocode of NOSI's implementation of NOSA inference in Appendix A.

We build NOSI with *kernel fusion*, specialized *memory layouts* and *block-selection* strategies, and custom kernels for *offloading communication*. Implementation details are provided in Appendix D.

4. Experiments

To examine NOSA's performance and efficiency, we conduct an extensive benchmark to address the following questions:
(**Q1**) Does NOSA achieve better performance on long-context inputs compared with other baselines?
(**Q2**) Does NOSA deliver higher decoding efficiency?

We also include additional results on general short-context tasks and reasoning tasks (App. E.1), length generalization (App. E.3) and ablation studies (App. E.5).

4.1. Experimental Setup

We briefly describe the major experimental settings here. Additional details can be found in Appendix B.

Datasets. We select representative benchmark datasets covering long-context input, long-generation, general tasks, and efficiency evaluation. Please refer to Appendix B.3 for tested datasets and Appendix B.4 for detailed settings.

Table 1. ***LongBench***[1] and ***Helmet*** scores (%) of NOSA compared with all baselines. NOSA^F uses full attention during the prefill stage and sparse attention during the decode stage, while NOSA^S adopts sparse attention for both stages. 1B and 3B results are in Table 11.

Method	*LongBench*														*Helmet*							
	GO	TQ	NQ	QS	MU	2W	MQ	RB	HQ	TR	PR	PC	SA	Avg. ↑	Recall	RAG	ICL	Cite	Rerank	LongQA	Summ.	Avg. ↑
8B Model Full Prefill																						
FULLATTN	31.0	88.1	19.4	25.0	27.4	40.6	52.3	58.3	50.0	74.5	94.0	3.0	6.6	43.9	$89.5_{\pm 0.3}$	$56.1_{\pm 1.0}$	$61.3_{\pm 0.4}$	$14.5_{\pm 0.4}$	$39.8_{\pm 0.0}$	$26.6_{\pm 0.6}$	$10.7_{\pm 1.0}$	$42.6_{\pm 0.3}$
SHADKV$_{64}$	28.5	87.8	**19.4**	24.2	25.0	41.1	52.0	57.4	48.7	**74.5**	89.5	3.0	**6.2**	42.9	$55.3_{\pm 0.2}$	$55.6_{\pm 0.9}$	$58.9_{\pm 0.6}$	$10.7_{\pm 0.6}$	$21.3_{\pm 0.1}$	$25.2_{\pm 1.2}$	$7.8_{\pm 0.5}$	$33.5_{\pm 0.3}$
SHADKV$_{8}$	28.8	87.7	**19.4**	24.4	26.2	**41.4**	51.4	57.5	48.8	**74.5**	**93.0**	3.0	6.1	43.3	$78.3_{\pm 0.4}$	$55.7_{\pm 1.1}$	$60.6_{\pm 0.5}$	$12.4_{\pm 0.9}$	$29.3_{\pm 0.0}$	$26.6_{\pm 0.8}$	$\mathbf{8.8}_{\pm 1.3}$	$38.8_{\pm 0.6}$
ARKVALE	29.7	**88.4**	17.6	24.6	25.0	33.7	36.2	45.9	42.7	71.0	92.5	3.0	3.4	39.5	$56.2_{\pm 1.5}$	$55.6_{\pm 0.8}$	$61.5_{\pm 0.6}$	$10.7_{\pm 2.4}$	$\mathbf{35.1}_{\pm 0.1}$	$26.5_{\pm 1.5}$	$8.2_{\pm 0.8}$	$36.2_{\pm 0.4}$
DMA^F	27.6	86.7	16.7	23.5	24.4	39.0	47.1	55.1	47.8	73.0	69.0	**6.0**	5.7	40.1	$33.1_{\pm 0.2}$	$52.4_{\pm 0.6}$	$\mathbf{63.8}_{\pm 0.1}$	$10.4_{\pm 0.7}$	$26.5_{\pm 0.0}$	$25.7_{\pm 1.9}$	$6.5_{\pm 0.7}$	$31.2_{\pm 0.5}$
NOSA^F	**31.8**	86.8	15.2	**25.1**	**29.2**	41.1	**53.0**	**59.7**	**50.7**	74.0	92.0	4.0	5.4	**43.7**	$\mathbf{86.3}_{\pm 0.0}$	$\mathbf{56.3}_{\pm 0.4}$	$60.6_{\pm 0.5}$	$\mathbf{13.6}_{\pm 1.1}$	$30.8_{\pm 0.0}$	$\mathbf{26.7}_{\pm 1.5}$	$8.5_{\pm 0.7}$	$\mathbf{40.4}_{\pm 0.5}$
8B Model Sparse Prefill																						
INFLLMV2	32.4	86.6	17.7	25.0	29.3	42.2	52.9	59.5	48.7	75.5	95.5	4.5	5.8	44.3	$72.7_{\pm 0.2}$	$54.1_{\pm 1.1}$	$68.9_{\pm 0.3}$	$13.8_{\pm 0.7}$	$37.5_{\pm 0.0}$	$26.6_{\pm 1.5}$	$8.6_{\pm 0.6}$	$40.3_{\pm 0.3}$
SHADKV$^M_{64}$	28.3	89.5	18.1	24.0	23.6	37.8	**52.9**	56.6	**49.0**	72.5	74.5	3.0	4.9	41.1	$54.5_{\pm 0.9}$	$53.1_{\pm 0.7}$	$52.4_{\pm 0.5}$	$10.7_{\pm 1.1}$	$23.4_{\pm 0.0}$	$23.9_{\pm 1.1}$	$8.1_{\pm 0.6}$	$32.3_{\pm 0.4}$
SHADKV$^M_{8}$	28.3	**89.6**	17.4	23.8	24.3	37.6	52.4	56.7	48.6	72.5	75.5	2.5	4.8	41.1	$\mathbf{74.6}_{\pm 0.2}$	$53.1_{\pm 0.7}$	$53.5_{\pm 0.2}$	$11.6_{\pm 0.5}$	$28.9_{\pm 0.0}$	$25.9_{\pm 2.1}$	$7.7_{\pm 1.0}$	$36.5_{\pm 0.4}$
INFLLM$_{128}$	30.4	87.0	18.7	22.8	31.8	40.7	50.1	**60.6**	48.2	71.5	49.0	**7.5**	3.4	40.1	$5.8_{\pm 0.1}$	$47.2_{\pm 0.5}$	$36.7_{\pm 0.9}$	$1.7_{\pm 0.1}$	$2.2_{\pm 0.0}$	$16.6_{\pm 2.5}$	$8.1_{\pm 0.4}$	$16.9_{\pm 0.3}$
INFLLM$_{64}$	30.2	86.2	**19.0**	23.1	**33.8**	38.8	50.2	60.4	**49.0**	**75.0**	57.5	5.5	3.9	41.0	$5.5_{\pm 0.5}$	$47.8_{\pm 1.2}$	$37.4_{\pm 0.8}$	$1.4_{\pm 0.5}$	$2.4_{\pm 0.0}$	$13.5_{\pm 3.9}$	$7.0_{\pm 1.2}$	$16.4_{\pm 0.7}$
DMA^S	27.9	87.2	15.0	23.1	17.9	36.6	45.1	53.9	44.4	73.0	35.5	2.5	4.4	35.9	$22.7_{\pm 0.7}$	$39.9_{\pm 0.7}$	$56.1_{\pm 0.4}$	$9.3_{\pm 0.9}$	$21.6_{\pm 0.0}$	$\mathbf{28.2}_{\pm 1.8}$	$6.5_{\pm 0.8}$	$26.3_{\pm 0.3}$
NOSA^S	**32.6**	86.8	16.5	**24.4**	29.1	40.8	**52.9**	59.0	48.3	74.0	**89.0**	4.0	**5.2**	**43.3**	$67.2_{\pm 0.8}$	$\mathbf{54.3}_{\pm 1.2}$	$\mathbf{65.1}_{\pm 0.5}$	$\mathbf{12.5}_{\pm 1.3}$	$\mathbf{31.0}_{\pm 0.0}$	$25.1_{\pm 1.8}$	$\mathbf{8.2}_{\pm 0.7}$	$\mathbf{37.6}_{\pm 0.7}$

Models. We evaluate three models (1B/3B/8B) which are pretrained at 4K context length then extended to 16K with NOSA ($k = 4096, k_q = 1024$) by long-context continual pretraining, followed by SFT. INFLLMV2 and DMA use the same pipeline. More details are in Appendix B.2.1.

Baselines. INFLLMV2 and FULLATTN are treated as the upper bound of our method. We include SHADOWKV (abbreviated as SHADKV), INFLLM, and ARKVALE as our primary training-free offloading baselines. We also include VLLM and SGLANG in the efficiency benchmark. Please refer to Appendix B.2.2 and C for detailed settings.

4.2. Long Context Tasks

To answer **Q1**, we evaluate NOSA against the baselines on LongBench and HELMET (Table 1).

NOSA achieves the highest performance on long-context inputs compared with all baselines. Table 1 shows that NOSA achieves consistently strong task performance compared to other baselines under both full prefill and sparse prefill settings, with no significant degradation relative to INFLLMV2 and FULLATTN. In contrast, ARKVALE and INFLLM perform poorly on HELMET, particularly in the sparse-prefill setting. SHADOWKV achieves relatively better performance, but still falls short of NOSA.

NOSA excels in hard context retrieval tasks. For the baselines, introducing sparsity patterns unseen during training severely degrades their ability to handle hard retrieval tasks. For example, all baselines suffer a substantial performance drop ($\geq 10\%$) on the PR task in LongBench (sparse prefill) and the Recall task in HELMET (full prefill). In contrast, NOSA maintains consistent performance and exhibits only a modest degradation compared to other baselines. Notably, due to DMA's inability to recall evicted context by design, it exhibits low performance on both overall and recall tasks.

[1]See Appendix B.3 for *LongBench* task abbreviations.

4.3. Efficiency Benchmarks

To address **Q2**, we compare NOSA with baseline methods in terms of decoding throughput and evaluate different implementations to demonstrate the effectiveness of NOSI.

Larger batch sizes improve decoding efficiency. Table 2 shows that offloading methods (SHADOWKV, NOSA) achieve higher throughput than non-offloading baselines by enabling larger batch sizes under the same memory budget. NOSA reaches up to 5.04× higher decoding throughput than FULLATTN at 96K input length and EB 64, as its real batch size is 16× larger when matching on-GPU KV size.

NOSA achieves the highest decoding throughput. Among offloading-based approaches, NOSA consistently delivers the best decoding throughput. ARKVALE does not utilize grouped-query attention (GQA)[2] for our model architecture, resulting in a smaller achievable batch size and consequently limited throughput. INFLLM suffers from an inefficient implementation, leading to low decoding throughput and high GPU memory utilization. SHADOWKV is the most competitive baseline; nevertheless, NOSA achieves up to 1.83× higher throughput than SHADOWKV.

Higher locality leads to faster decoding speed. Since NOSA explicitly incorporates locality constraints, it achieves better locality and less CPU-GPU communication than INFLLMV2. Consequently, NOSA attains up to a 1.92× higher decoding throughput over INFLLMV2.

Implementation matters. Under HF, NOSA is slower than INFLLMV2 due to suboptimal kernels and excessive small kernel launches. However, under NOSI, the overheads are largely mitigated, revealing communication as the true bottleneck; consequently, NOSA surpasses all baselines.

[2]ARKVALE only supports GQA with a q:k ratio of 1:1, 4:1, or 8:1. For our model (16:1), the official impl. degrades to MHA.

Table 2. Decoding throughput (tok/s) ↑. "EB" = equivalent batch size; "Impl."/"Off." = implementation/offloading. For each EB, we fix GPU KV cache size and report the real batch size in parentheses. "–" indicates no feasible batch (≥ 1); "OOM" = out-of-memory.

Method	Impl.	Off.	*Input Length: 16K*				Method	Impl.	Off.	*Input Length: 32K*			
			EB: 16	*EB: 32*	*EB: 64*	*EB: 128*				*EB: 16*	*EB: 32*	*EB: 64*	*EB: 128*
FullAttn	HF	✗	160.85 (4)	239.94 (8)	OOM (16)	OOM (32)	FullAttn	HF	✗	80.56 (2)	120.99 (4)	OOM (8)	OOM (16)
FullAttn	vLLM	✗	233.21 (4)	417.99 (8)	678.41 (16)	927.99 (32)	FullAttn	vLLM	✗	113.42 (2)	151.58 (4)	460.65 (8)	818.53 (16)
FullAttn	SGLang	✗	180.28 (4)	358.56 (8)	774.74 (16)	1267.20 (32)	FullAttn	SGLang	✗	116.54 (2)	174.50 (4)	290.29 (8)	748.20 (16)
InfLLMv2	HF	✗	47.10 (4)	OOM (8)	OOM (16)	OOM (32)	InfLLMv2	HS	✗	23.17 (2)	OOM (4)	OOM (8)	OOM (16)
InfLLMv2	NOSI	✗	226.51 (4)	440.76 (8)	824.09 (16)	1475.43 (32)	InfLLMv2	NOSI	✗	104.88 (2)	205.74 (4)	403.26 (8)	710.93 (16)
InfLLMv2	NOSI	✓	563.88 (16)	759.35 (32)	1389.40 (64)	1441.32 (128)	InfLLMv2	NOSI	✓	491.42 (16)	623.50 (32)	1132.50 (64)	652.60 (128)
ShadKV	ShadKV	✓	585.92 (16)	769.84 (32)	970.19 (64)	1071.36 (128)	ShadKV	ShadKV	✓	541.91 (16)	697.10 (32)	885.05 (64)	995.81 (128)
ArkVale	ArkVale	✓	34.15 (1)	46.76 (2)	57.38 (4)	68.65 (8)	ArkVale	ArkVale	✓	29.45 (1)	38.15 (2)	45.05 (4)	OOM (8)
InfLLM	InfLLM	✓	35.00 (16)	OOM (32)	OOM (64)	OOM (128)	InfLLM	InfLLM	✓	30.72 (16)	OOM (32)	OOM (64)	OOM (128)
NOSA	HF	✗	14.69 (4)	OOM (8)	OOM (16)	OOM (32)	NOSA	HF	✗	3.62 (2)	OOM (4)	OOM (8)	OOM (16)
NOSA	NOSI	✗	206.10 (4)	411.72 (8)	817.12 (16)	1503.76 (32)	NOSA	NOSI	✗	96.15 (2)	205.18 (4)	402.82 (8)	772.19 (16)
NOSA	**NOSI**	✓	**743.18** (16)	**1056.27** (32)	**1630.80** (64)	**1961.17** (128)	**NOSA**	**NOSI**	✓	**694.58** (16)	**1067.32** (32)	**1574.47** (64)	**1536.19** (128)

Method	Impl.	Off.	*Input Length: 64K*				Method	Impl.	Off.	*Input Length: 96K*			
			EB: 8	*EB: 16*	*EB: 32*	*EB: 64*				*EB: 8*	*EB: 16*	*EB: 32*	*EB: 64*
FullAttn	HF	✗	–	40.80 (1)	60.41 (2)	OOM (4)	FullAttn	HF	✗	–	35.17 (1)	48.67 (2)	OOM (4)
FullAttn	vLLM	✗	–	60.22 (1)	82.40 (2)	240.28 (4)	FullAttn	vLLM	✗	–	48.48 (1)	120.76 (2)	206.90 (4)
FullAttn	SGLang	✗	–	45.19 (1)	81.60 (2)	191.65 (4)	FullAttn	SGLang	✗	–	47.23 (1)	78.47 (2)	214.42 (4)
InfLLMv2	HF	✗	–	11.14 (1)	OOM (2)	OOM (4)	InfLLMv2	HS	✗	–	9.26 (1)	OOM (2)	OOM (4)
InfLLMv2	NOSI	✗	–	46.62 (1)	89.77 (2)	174.66 (4)	InfLLMv2	NOSI	✗	–	40.05 (1)	78.06 (2)	153.15 (4)
InfLLMv2	NOSI	✓	273.54 (8)	409.34 (16)	561.95 (32)	717.65 (64)	InfLLMv2	NOSI	✓	264.92 (8)	506.80 (16)	444.61 (32)	725.26 (64)
ShadKV	ShadKV	✓	292.97 (8)	503.73 (16)	686.69 (32)	866.01 (64)	ShadKV	ShadKV	✓	271.40 (8)	457.08 (16)	625.71 (32)	730.48 (64)
ArkVale	ArkVale	✓	–	25.42 (1)	32.83 (2)	OOM (4)	ArkVale	ArkVale	✓	–	24.43 (1)	30.35 (2)	OOM (4)
InfLLM	InfLLM	✓	30.04 (8)	OOM (16)	OOM (32)	OOM (64)	InfLLM	InfLLM	✓	24.96 (8)	OOM (16)	OOM (32)	OOM (64)
NOSA	HF	✗	–	0.81 (1)	OOM (2)	OOM (4)	NOSA	HF	✗	–	0.27 (1)	OOM (2)	OOM (4)
NOSA	NOSI	✗	–	46.08 (1)	84.91 (2)	176.23 (4)	NOSA	NOSI	✗	–	38.08 (1)	78.50 (2)	154.11 (4)
NOSA	**NOSI**	✓	**312.39** (8)	**545.08** (16)	**763.82** (32)	**1378.82** (64)	**NOSA**	**NOSI**	✓	**283.44** (8)	**542.03** (16)	**666.59** (32)	**1080.45** (64)

5. Related Works

Here, we briefly review sparse attention and offloading. Other KV cache optimizations are discussed in Appendix C.

Sparse Attention. Sparse and approximate attention mechanisms have been proposed attention's dense computation and memory access. Early approaches (Xiao et al., 2024b; Han et al., 2024b) employ simple and fixed sparsity patterns. More advanced methods (Ge et al., 2024; Jiang et al., 2024; Xu et al., 2025; Xiao et al., 2025; Zhang et al., 2025a;d) introduce richer or more dynamic sparsity structures for improved task performance, often combined with other optimizations like low-bit quantization (Zhang et al., 2025b;c; Yang et al., 2024). Among these methods, query-aware sparsification is a widely adopted strategy (Li et al., 2024; Zhang et al., 2023b; Liu et al., 2023). In contrast, query-agnostic methods (Huang et al., 2025; Kim et al., 2024; Yao et al., 2024) focus on selecting essential KV pairs without queries, but suffer from larger performance degradation.

Recent studies (DeepSeek-AI, 2025a; Shi et al., 2025; Yan et al., 2025; Vasylenko et al., 2025; Gonçalves et al., 2025) focus on eliminating the mismatch between training and inference when applying sparse attentions, i.e., integrating sparsity during model training. NSA-like methods (Yuan et al., 2025; Gao et al., 2025; Lu et al., 2025) introduce trainable block-sparse patterns tailored for long-context scenarios, though these approaches tend to slow down short-context inference. InfLLMv2 (Zhao et al., 2025) addresses this limitation by designing dense-sparse switchable patterns for short and long contexts. Beyond LLMs, sparse attentions have also been applied to various domains (Zhang et al., 2025f; Wang et al., 2025; Yoshai et al., 2025).

LLM Offloading Systems. Offloading is a widely adopted technique for LLM inference in memory-constrained environments. Early approaches (Sheng et al., 2023; Song et al., 2024) focus on parameter offloading to inference larger LLMs on smaller GPUs. Block-wise KV offloading strategies are proposed for handling longer input sequences (Tang et al., 2024; Xiao et al., 2024a; Chen et al., 2024a) and increasing decoding throughputs (Sun et al., 2025; Chen et al., 2024b). More recent studies (Yang et al., 2025; Zhou et al., 2025; Chen et al., 2025; Liu et al., 2025; Wu et al., 2025) focusing on supporting large-scale serving systems with sparse attentions to enhance their decoding efficiency.

6. Conclusion

We propose NOSA, a trainable sparse attention mechanism for KV-cache offloading with an explicit locality constraint, and implement an inference system NOSI to show its efficiency. We evaluate performance and decoding efficiency on 1B, 3B, and 8B models. NOSA outperforms KV cache offloading baselines on general, long-context input, and long-generation tasks, while improving decoding throughput by up to $5.04\times$, $1.92\times$, and $1.83\times$ over FullAttn, InfLLMv2, and ShadowKV. Future work includes integrating NOSA and NOSI into LLM serving engines and accelerating the rollout in LLM reinforcement learning.

Impact Statement

This paper aims to advance the field of machine learning. We expect our method to improve LLM decoding efficiency, which can lower the compute and energy cost of inference. By making long-context decoding more efficient, our approach may facilitate broader deployment of LLM applications while reducing their environmental footprint. We do not foresee any potential negative ethical or societal impacts arising from this work.

References

Anthropic. Claude-sonnet-4.5, 2025.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

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 ACL*, pp. 3119–3137, 2024.

Bhaskar, A., Wettig, A., Gao, T., Dong, Y., and Chen, D. Cache me if you can: How many kvs do you need for effective long-context lms? *arXiv preprint arXiv:2506.17121*, 2025.

Cao, Z., Si, Q., Zhang, J., and Liu, B. Sparse attention across multiple-context kv cache. *arXiv preprint arXiv:2508.11661*, 2025.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chen, R., Wang, Z., Cao, B., Wu, T., Zheng, S., Li, X., Wei, X., Yan, S., Li, M., and Liang, Y. Arkvale: Efficient generative llm inference with recallable key-value eviction. *Proceedings of NeurIPS*, 2024a.

Chen, X., Zhang, C., He, J., Liu, W., Zhang, J., Zhou, W., Li, X., Zeng, P., Li, S., Qian, Y., et al. Ess: An offload-centric latent-cache management architecture for deepseek-v3. 2-exp. *arXiv preprint arXiv:2512.10576*, 2025.

Chen, Z., Sadhukhan, R., Ye, Z., Zhou, Y., Zhang, J., Nolte, N., Tian, Y., Douze, M., Bottou, L., Jia, Z., et al. Magicpig: Lsh sampling for efficient llm generation. *Proceedings of ICLR*, 2024b.

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.

DeepSeek-AI. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention, 2025a.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Nature*, 2025b.

Ding, Y., Zhang, L. L., Zhang, C., Xu, Y., Shang, N., Xu, J., Yang, F., and Yang, M. Longrope: Extending llm context window beyond 2 million tokens. *Proceedings of ICML*, 2024.

Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., and Gardner, M. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *Proceedings of ACL*, 2019.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Gao, Y., Zeng, Z., Du, D., Cao, S., Zhou, P., Qi, J., Lai, J., So, H. K.-H., Cao, T., Yang, F., et al. Seerattention: Learning intrinsic sparse attention in your llms. *Proceedings of NeurIPS*, 2025.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms. *Proceedings of ICLR*, 2024.

Gonçalves, N., Treviso, M., and Martins, A. F. Adasplash: Adaptive sparse flash attention. *Proceedings of ICML*, 2025.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Guo, M.-H., Xu, J., Zhang, Y., Song, J., Peng, H., Deng, Y.-X., Dong, X., Nakayama, K., Geng, Z., Wang, C., et al. R-bench: Graduate-level multi-disciplinary benchmarks for

llm & mllm complex reasoning evaluation. *Proceedings of ICML*, 2025.

Han, C., Wang, Q., Peng, H., Xiong, W., Chen, Y., Ji, H., and Wang, S. Lm-infinite: Zero-shot extreme length generalization for large language models. *Proceedings of ACL*, 2024a.

Han, C., Wang, Q., Xiong, W., Chen, Y., Ji, H., and Wang, S. Lm-infinite: Simple on-the-fly length generalization for large language models. *Proceedings of NAACL*, 2024b.

He, X., Liu, J., and Chen, S. Task-kv: Task-aware kv cache optimization via semantic differentiation of attention heads. *arXiv preprint arXiv:2501.15113*, 2025.

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. *Proceedings of NeurIPS*, 2021.

Hooper, C., Kim, S., Mohammadzadeh, H., Mahoney, M. W., Shao, Y. S., Keutzer, K., and Gholami, A. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Proceedings of NeurIPS*, 2024.

Hooper, C. R. C., Kim, S., Mohammadzadeh, H., Maheswaran, M., Zhao, S., Paik, J., Mahoney, M. W., Keutzer, K., and Gholami, A. Squeezed attention: Accelerating long context length llm inference. *Proceedings of ACL*, 2025.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. Ruler: What's the real context size of your long-context language models? *Proceedings of COLM*, 2024.

Hu, S., Tu, Y., Han, X., He, C., Cui, G., Long, X., Zheng, Z., Fang, Y., Huang, Y., Zhao, W., et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *Proceedings of COLM*, 2024.

Huang, Y., Yuan, B., Han, X., Xiao, C., and Liu, Z. Locret: Enhancing eviction in long-context llm inference with trained retaining heads on consumer-grade devices. *Proceedings of TMLR*, 2025.

Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Ji, T., Guo, B., Wu, Y., Guo, Q., Shen, L., Chen, Z., Qiu, X., Zhang, Q., and Gui, T. Towards economical inference: Enabling deepseek's multi-head latent attention in any transformer-based llms. *Proceedings of ACL*, 2025.

Jiang, H., Li, Y., Zhang, C., Wu, Q., Luo, X., Ahn, S., Han, Z., Abdi, A. H., Li, D., Lin, C.-Y., et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Proceedings of NeurIPS*, 2024.

Jiang, Z., Li, K., Zhou, Y., Liu, S., Wang, Z., Zhang, S., et al. Purekv: Plug-and-play kv cache optimization with spatial-temporal sparse attention for vision-language large models. *arXiv preprint arXiv:2510.25600*, 2025.

Kim, M., Shim, K., Choi, J., and Chang, S. Infinipot: Infinite context processing on memory-constrained llms. *Proceedings of EMNLP*, 2024.

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. *Proceedings of NeurIPS*, 2024.

Li, Y., Jiang, H., Zhang, C., Wu, Q., Luo, X., Ahn, S., Abdi, A. H., Li, D., Gao, J., Yang, Y., et al. Mminference: Accelerating pre-filling for long-context vlms via modality-aware permutation sparse attention. *Proceedings of ICML*, 2025.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *Proceedings of ICLR*, 2024.

Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Liu, A., Liu, J., Pan, Z., He, Y., Haffari, G., and Zhuang, B. Minicache: Kv cache compression in depth dimension for large language models. *Proceedings of NeurIPS*, 2024b.

Liu, G., Li, C., Zhao, J., Zhang, C., and Guo, M. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. *Proceedings of DAC*, 2025.

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. *Proceedings of NeurIPS*, 2023.

Liu, Z., Yuan, J., Jin, H., Zhong, S., Xu, Z., Braverman, V., Chen, B., and Hu, X. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *Proceedings of ICML*, 2024c.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *Proceedings of ICLR*, 2019.

Lu, E., Jiang, Z., Liu, J., Du, Y., Jiang, T., Hong, C., Liu, S., He, W., Yuan, E., Wang, Y., et al. Moba: Mixture of block attention for long-context llms. *Proceedings of NeurIPS*, 2025.

Luo, J., Zhang, W., Yuan, Y., Zhao, Y., Yang, J., Gu, Y., Wu, B., Chen, B., Qiao, Z., Long, Q., et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv:2503.21460*, 2025.

MiniCPM-Team. Minicpm4: Ultra-efficient llms on end devices. *arXiv preprint arXiv:2506.07900*, 2025.

Mu, J., Huang, H., Zhang, J., Yu, M., Wang, T., and Li, Y. Sals: Sparse attention in latent space for kv cache compression. *Proceedings of NeurIPS*, 2025.

OpenAI. Gpt-5, 2025.

OpenBMB. Infllm-v2-data-5b dataset. https://huggingface.co/datasets/openbmb/InfLLM-V2-data-5B, 2025.

Qin, Y., Hu, S., Lin, Y., Chen, W., Ding, N., Cui, G., Zeng, Z., Zhou, X., Huang, Y., Xiao, C., et al. Tool learning with foundation models. *ACM Computing Surveys*, 2024.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *Proceedings of ICLR*, 2020.

Saxena, U., Saha, G., Choudhary, S., and Roy, K. Eigen attention: Attention in low-rank space for kv cache compression. *Proceedings of EMNLP findings*, 2024.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single gpu. *Proceedings of ICML*, 2023.

Shi, J., Wu, Y., Wu, B., Peng, Y., Wang, L., Liu, G., and Luo, Y. Trainable dynamic mask sparse attention. *arXiv preprint arXiv:2508.02124*, 2025.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Singhania, P., Singh, S., He, S., Feizi, S., and Bhatele, A. Loki: Low-rank keys for efficient sparse attention. *Proceedings of NeurIPS*, 2024.

Song, Y., Mi, Z., Xie, H., and Chen, H. Powerinfer: Fast large language model serving with a consumer-grade gpu. *Proceedings of SOSP*, 2024.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Sun, H., Chang, L.-W., Bao, W., Zheng, S., Zheng, N., Liu, X., Dong, H., Chi, Y., and Chen, B. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *Proceedings of ICML*, 2025.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., , and Wei, J. Challenging big-bench tasks and whether chain-of-thought can solve them. *Proceedings of ACL Findings*, 2023.

Tang, J., Zhao, Y., Zhu, K., Xiao, G., Kasikci, B., and Han, S. Quest: Query-aware sparsity for efficient long-context llm inference. *Proceedings of ICML*, 2024.

Tao, Q., Yu, W., and Zhou, J. Asymkv: Enabling 1-bit quantization of kv cache with layer-wise asymmetric quantization configurations. *Proceedings of ACL*, 2025.

Vasylenko, P., Pitorro, H., Martins, A. F., and Treviso, M. Long-context generalization with sparse attention. *arXiv preprint arXiv:2506.16640*, 2025.

Wang, H., Liu, J., Chen, Z., Zhang, Y., and Li, C. Sparse attention diffusion model for pathological micrograph deblurring. *International Conference on Artificial Neural Networks*, 2025.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Proceedings of NeurIPS*, 2024a.

Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Proceedings of NeurIPS*, 2024b.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. *Proceedings EMNLP System Demonstrations*, 2020.

Wu, H. and Tu, K. Layer-condensed kv cache for efficient inference of large language models. *Proceedings of ACL*, 2024.

Wu, W., Si, Q., Pan, X., Wang, Y., and Zhang, J. Louiskv: Efficient kv cache retrieval for long input-output sequences. *arXiv preprint arXiv:2510.11292*, 2025.

Xiao, C., Zhang, P., Han, X., Xiao, G., Lin, Y., Zhang, Z., Liu, Z., and Sun, M. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Proceedings of NeurIPS*, 2024a.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *Proceedings of ICLR*, 2024b.

Xiao, G., Tang, J., Zuo, J., Guo, J., Yang, S., Tang, H., Fu, Y., and Han, S. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *Proceedings of ICLR*, 2025.

Xu, R., Xiao, G., Huang, H., Guo, J., and Han, S. Xattention: Block sparse attention with antidiagonal scoring. *Proceedings of ICML*, 2025.

Yan, R., Jiang, Y., and Yuan, B. Flash sparse attention: More efficient natively trainable sparse attention. *arXiv preprint arXiv:2508.18224*, 2025.

Yang, G., Hu, E., Babuschkin, I., Sidor, S., Liu, X., Farhi, D., Ryder, N., Pachocki, J., Chen, W., and Gao, J. Tuning large neural networks via zero-shot hyperparameter transfer. *Proceedings of NeurIPS*, 2021.

Yang, S., Sheng, Y., Gonzalez, J. E., Stoica, I., and Zheng, L. Post-training sparse attention with double sparsity. *arXiv preprint arXiv:2408.07092*, 2024.

Yang, S., Guo, J., Tang, H., Hu, Q., Xiao, G., Tang, J., Lin, Y., Liu, Z., Lu, Y., and Han, S. Lserve: Efficient long-sequence llm serving with unified sparse attention. *Proceedings of MLSys*, 2025.

Yao, Y., Li, Z., and Zhao, H. Sirllm: Streaming infinite retentive llm. *Proceedings of ACL*, 2024.

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. *Proceedings of MLSys*, 2025.

Yen, H., Gao, T., Hou, M., Ding, K., Fleischer, D., Izsak, P., Wasserblat, M., and Chen, D. Helmet: How to evaluate long-context language models effectively and thoroughly. *Proceedings of ICLR*, 2025.

Yoshai, E., Yagoda-Aharoni, D., Dotan, E., and Shaked, N. T. Hierarchical sparse attention framework for computationally efficient classification of biological cells. *arXiv preprint arXiv:2505.07661*, 2025.

Yuan, J., Gao, H., Dai, D., Luo, J., Zhao, L., Zhang, Z., Xie, Z., Wei, Y., Wang, L., Xiao, Z., et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *Proceedings of ACL*, 2025.

Yuan, Z., Shang, Y., Zhou, Y., Dong, Z., Xue, C., Wu, B., Li, Z., Gu, Q., Lee, Y. J., Yan, Y., Chen, B., Sun, G., and Keutzer, K. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

Zandieh, A., Daliri, M., and Han, I. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead. *Proceedings of AAAI*, 2025.

Zhang, J., Huang, H., Zhang, P., Wei, J., Zhu, J., and Chen, J. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization. In *Proceedings of ICML*, 2025a.

Zhang, J., Wei, J., Zhang, P., Xu, X., Huang, H., Wang, H., Jiang, K., Zhu, J., and Chen, J. Sageattention3: Microscaling fp4 attention for inference and an exploration of 8-bit training. *Proceedings of NeurIPS*, 2025b.

Zhang, J., Wei, J., Zhang, P., Zhu, J., and Chen, J. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *Proceedings of ICLR*, 2025c.

Zhang, J., Xu, X., Wei, J., Huang, H., Zhang, P., Xiang, C., Zhu, J., and Chen, J. Sageattention2++: A more efficient implementation of sageattention2. *arXiv preprint arXiv:2505.21136*, 2025d.

Zhang, M., Sun, H., Wang, J., Li, S., Ning, W., Qi, Q., Zhuang, Z., and Liao, J. Clusterattn: Kv cache compression under intrinsic attention clustering. *Proceedings of ACL*, 2025e.

Zhang, P., Chen, Y., Huang, H., Lin, W., Liu, Z., Stoica, I., Xing, E., and Zhang, H. Vsa: Faster video diffusion with trainable sparse attention. *Proceedings of NeurIPS*, 2025f.

Zhang, X., Li, C., Zong, Y., Ying, Z., He, L., and Qiu, X. Evaluating the performance of large language models on gaokao benchmark. *arXiv preprint arXiv:2305.12474*, 2023a.

Zhang, X., Zhou, Y., Yang, G., Gall, H. C., and Chen, T. Anchor attention, small cache: Code generation with large language models. *IEEE Transactions on Software Engineering*, 2025g.

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. *Proceedings of NeurIPS*, 2023b.

Zhao, W., Zhou, Z., Su, Z., Xiao, C., Li, Y., Li, Y., Zhang, Y., Zhao, W., Li, Z., Huang, Y., et al. Infllm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation. *arXiv preprint arXiv:2509.24663*, 2025.

Zhou, Q., Yin, P., Zuo, P., and Cheng, J. Sparseserve: Unlocking parallelism for dynamic sparse attention in long-context llm serving. *arXiv preprint arXiv:2509.24626*, 2025.

A. Pseudocode of NOSA

Algorithm 1 NOSA's attention calculation (NOSI implementation) at the t-th decoding step.

Input: Hidden state $\mathbf{H} \in \mathbb{R}^{B\times 1\times d}$, Linear/eviction head weights $\mathbf{W}_{\text{qkv}}, \mathbf{W}_1, \mathbf{W}_2$, Cache engine **engine**, Hyperparameters **params** including k, k_e, k_q, block size, pooling strides, etc.
Output: Attention output $\mathbf{H}' \in \mathbb{R}^{B\times 1\times d}$.

$\mathbf{Q}, \mathbf{K}, \mathbf{V}, s_t^e =$ `fused_linear_projection`$(\mathbf{H}, \mathbf{W}_{\text{qkv}}, \mathbf{W}_1, \mathbf{W}_2)$
`apply_rope_inplace`$(\mathbf{Q}, \mathbf{K})$

$\mathbf{K_c}$ = **cache_engine**.`update_compressed_k`$(\mathbf{K})$
$\mathbf{S}_c^e, \mathbf{S}^e$ = **cache_engine**.`update_importance_score`(s_t^e)

// Stage 1: Select Top-k blocks to attend
$\mathbf{S}_c^q = \mathbf{Q}\mathbf{K}_c^\top$
q_aware_score, **q_agnostic_score** = `fused_pooling`$(\mathbf{S}_c^q, \mathbf{S}_c^e,$ **params**$)$

// Select up to k_q tokens from **q_aware_score** to maintain a minimum locality of k_e/k
topk_idx = `select_topk_indices`(**q_aware_score**, **q_agnostic_score**, **params**.k_e, **params**.k)

// Move cache-missed KV blocks from CPU to GPU
load_pos = `generate_load_positions`(**topk_idx**, **engine.current_mapping**)
engine.`host_to_device_communication`(**load_pos**)

// Stage 2: The real attention calculation
$\mathbf{H}'$ = `flash_attn_with_kvcache`($\mathbf{Q}$, **cache.key**, **cache.value**)
return $\mathbf{H}'$

B. Reproducibility Settings

B.1. Environment

We conduct training on 32 NVIDIA A800 GPUs across four nodes (8 GPUs per node). Intra-node GPU communication leverages third-generation NVLink, while inter-node communication uses HDR InfiniBand. Each node is equipped with Intel® Xeon® Platinum 8470 CPUs and ran CentOS Linux 7 (Core). Each node comprises two NUMA domains with 52 CPU cores in total, and is provisioned with 1 TB of DDR5 memory configured at 4400 MT/s. GPU–CPU communication is conducted over a PCIe 4.0×16 interface, with a peak bandwidth of 31.5 GB/s. All models are trained using the Megatron framework (Shoeybi et al., 2019). For all settings in the inference experiments, we use a single GPU and 26 CPU cores from the NUMA domain closest to the GPU on the same cluster. All methods except ARKVALE use `bfloat16` precision in both training and inference, while ARKVALE uses `float16` precision since ARKVALE does not have `bfloat16` kernels.

B.2. Hyperparameters

B.2.1. MODEL TRAINING

We train 1B, 3B, and 8B models under the pipeline illustrated in Figure 3 for FULLATTN, INFLLMV2, DMA, and the NOSA variant, resulting in a total of 12 models. The model architectures and training configurations are summarized in Table 3. For the 1B and 3B settings, we adopt the Llama-3 architecture (Grattafiori et al., 2024), while we take MiniCPM4-base (MiniCPM-Team, 2025) as our 8B base models. We apply LongRoPE (Ding et al., 2024) to extend the 4K context length in the pretraining stage to 16K in the long-context continual pretraining stage. All models share the MiniCPM4 tokenizer. We use the AdamW optimizer (Loshchilov & Hutter, 2019) together with the WSD learning rate scheduler (Hu et al., 2024). Long-context continual pretraining is conducted on the InfLLMv2-data-5B dataset (OpenBMB, 2025).

B.2.2. HYPERPARAMETERS OF EACH METHOD

INFLLMV2. We adopt most hyperparameters from the official INFLLMV2 configuration. The mean-pooling stride l_s^{mean} is set to 32 and the mean-pooling kernel size l_k^{mean} is set to 16. We set the block size n_b to 64. For each query token, we attend to $k = 4096$ selected tokens, consisting of one block reserved for the attention sink, 16 blocks for the sliding window, and

Table 3. Model architecture and training details.

Model Size	1B	3B	8B
Architecture	Llama-3	Llama-3	MiniCPM4
Use μP (Yang et al., 2021)	✗	✗	✓
Vocabulary Size	73448	73448	73448
Number of Layers	28	32	32
Hidden Size	2048	2560	4096
Number of Attention Heads	16	32	32
Attention Head Dimention	128	128	128
Number of KV Heads	2	2	2
FFN Intermediate Size	6144	10240	16384
Position Embedding Type	LongRoPE	LongRoPE	LongRoPE
RoPE's (Su et al., 2024) θ	10000	10000	10000
Pretraining Context Length	4096	4096	4096
Long-context Continual Pretraining Context Length	16384	16384	16384
Optimizer	AdamW	AdamW	AdamW
LR-Scheduler	WSD	WSD	WSD
Long-context Continual Pretraining Starting LR	1.50e-5	1.50e-5	3.00e-4
Long-context Continual Pretraining Ending LR	1.34e-5	8.62e-6	2.00e-4
SFT Starting LR	7.50e-6	7.50e-6	2.00e-4
SFT Ending LR	0.00	0.00	0.00
Pretraining Tokens	1.5T	3.8T	8.0T
Long-Context Training Tokens	2.0B	3.4B	5.0B
SFT Tokens	0.6B	1.3B	1.9B
Long-context Continual Pretraining GPU Hours	336	883	1496
SFT GPU Hours	78	190	377

47 dynamically selected blocks. This configuration is used for both training and inference.

DMA. We use the same configuration as INFLLMV2, except that the 47 dynamically selected blocks are chosen based on the importance scores predicted by the eviction head.

NOSA. We follow the INFLLMV2 configuration and set $k = 4096$, $k_q = 1024$, and $k_e = 3072$. Specifically, 16 blocks are selected via query-aware selection, while the remaining 31 blocks (out of the 47 dynamic blocks) are chosen via query-agnostic selection. Since both the attention sink and the sliding window are inherently query-agnostic, we also count them as part of query-agnostic selection, resulting in a total of 48 query-agnostic blocks. Therefore, $\gamma(t) \geq 75\%$.

SHADOWKV. We also select 4096 tokens for sparse attention in SHADOWKV, following the same procedure as above. We evaluate four SHADOWKV variants on long-context input benchmarks (LongBench and Helmet): SHADKV_8, SHADKV_{64}, SHADKV_8^M, and SHADKV_{64}^M, where the superscript ^{M} indicates that MINFERENCE (Jiang et al., 2024) is applied during the prefill stage. We set the truncation rank in SVD to 40 for all settings. For SHADKV_8 and SHADKV_8^M, we set the block size to 8, allocate 256 tokens to the sliding window and 768 tokens to outliers, and reserve the remaining 3072 tokens for dynamic selection. For SHADKV_{64} and SHADKV_{64}^M, we use the same token allocation while increasing the block size to 64. The two settings are included because the official SHADOWKV configuration uses a block size of 8, and we additionally evaluate a block size of 64 to enable a fair comparison with NOSA. For long-generation benchmarks (reasoning tasks), we use the SHADKV_8 setting. Note that the official implementation of SHADOWKV supports a maximum generation length of only 1024 tokens, which is incompatible with our setting of generating 8192 tokens. Moreover, the generated tokens are retained in GPU memory and treated as a lengthening sliding window, which leads to an unfair comparison: by the end of an 8K-token generation, SHADOWKV activates a 12K context, whereas other methods activate only 4K. To enable long generation for SHADOWKV while ensuring a fair comparison under a 4K budget, we implement a V cache offloading strategy and factorize the K cache using SVD bases computed during prefill for long-generation tasks. This allows SHADOWKV to operate within the same 4K context budget as the other methods. For the efficiency benchmark, we cannot use 3072 dynamic tokens because the official SHADOWKV offloading kernels do not support this setting (we run in simulation mode for performance evaluation, following the official setup). Therefore, we set the number of dynamic tokens to 2048, the sliding-window size to 1280, and keep 768 tokens for outliers. Note that this configuration favors SHADOWKV's decoding efficiency and disadvantages NOSA. Nevertheless, NOSA remains faster than SHADOWKV under

this setting, and thus outperforms SHADOWKV in terms of efficiency.

ARKVALE. We also select 4096 tokens for sparse attention in ARKVALE. The block size is set to 32, the attention sink size is set to 64, and the sliding window size is set to 1024 to ensure a fair comparison. Note that the official implementation of ARKVALE does not support `bfloat16` inference; therefore, we use `float16` instead. A block size of 64 is also not supported, so we use 32. This configuration favors ARKVALE's performance. Nevertheless, since NOSA achieves higher accuracy on the benchmarks than ARKVALE, it still outperforms ARKVALE. Note that ARKVALE does not support the GQA head ratio of 16:1 (query:key) in its official implementation, and thus falls back to MHA.

INFLLM. We retain 4096 tokens in INFLLM. We evaluate two variants, INFLLM_{64} and INFLLM_{128}, with block sizes set to 64 and 128, respectively. Since the original INFLLM design uses a block size of 128, we additionally include the 64-block setting for a fair comparison with NOSA. We set the sliding-window size to 1024 and reserve one block for the attention sink. We set the representative top-k to 4, the maximum cached blocks to the number of activated blocks, and the execution block size to 512.

B.3. Datasets

For long-context evaluation, we compare NOSA with baselines on LongBench (Bai et al., 2024) and HELMET (Yen et al., 2025). We use only English tasks with an average input length exceeding 16K in LongBench to ensure higher data quality. HELMET allows controllable context lengths, and we set the input length to 16K. Notably, HELMET's recall subset consists of four challenging retrieval tasks (MK2, MK3, MV from RULER (Hsieh et al., 2024), and JsonKV) which collectively highlight the ability to handle long-context retrieval. Notably, since some baselines do not apply sparsity during the prefill stage, we report results for full prefill and sparse prefill separately to ensure a fair comparison. For general tasks, we test on MMLU (Wang et al., 2024a), MMLU-Pro (Wang et al., 2024b), BBH (Suzgun et al., 2023), GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), DROP (Dua et al., 2019), MBPP (Austin et al., 2021), and HumanEval (Chen et al., 2021). For reasoning tasks, we report results on MATH-500 (Lightman et al., 2024) and the Gaokao benchmarks (Zhang et al., 2023a), including math (Gaokao-MS and Gaokao-MH for the science and humanities subsets) and physics (Gaokao-Phy). Since the models are not trained on reasoning traces, we employ a 2-shot prompt (22K tokens) to elicit the model's reasoning ability. We evaluate the decoding efficiency on PG19 (Rae et al., 2020) dataset.

Task Abbreviations. Due to space limitations, we use abbreviations in Table 1 and Table 4. We provide the mapping between task names and their corresponding abbreviations below. Task abbreviations of *LongBench*: Gov_Report (GO), Triviaqa (TQ), NarrativeQA (NQ), QMSum (QS), Musique (MU), 2wikimqa (2W), MultifiedQA_En (MQ), RepoBench-P (RB), HotpotQA (HQ), Trec (TR), Passage_Retrieval_En (PR), Passage_Count (PC), and SamSum (SA). Task abbreviations of *general tasks*: MMLU (MM), MMLU-Pro (MMP), BBH (BB), GSM8K (GS), MATH (MA), DROP (DR), MBPP (MB), and HumanEval (HE). For general tasks with short sequences, sparsity and attention bias are not applied.

B.4. Benchmark Settings

Long-Context Inputs. We follow the official LongBench and HELMET protocols to benchmark all methods. For LongBench, we use greedy decoding for every method and report per-task scores. For HELMET, the official implementation samples evaluation instances from a large data pool, with a random seed controlling the sampled subset. Same as the official setting, the generation of LongQA and Summ. task in HELMET is evaluated by `gpt-4o` (Hurst et al., 2024). Following the official protocol, we report the mean score and standard deviation for each task. We also use greedy decoding on HELMET.

General Tasks. We use LM-Evaluation-Harness (Gao et al., 2024) to evaluate general-purpose benchmarks. We report 5-shot results for MMLU and MMLU-Pro, 3-shot for BBH, 5-shot flexible-accuracy for GSM8K, 4-shot for MATH, 0-shot F1 for DROP, 3-shot for MBPP, and 0-shot results on the instruction version of HumanEval. To improve performance, we apply a chat template for BBH, GSM8K, DROP, and HumanEval, but do not apply it for other tasks.

Reasoning Tasks. MiniCPM4-base is not trained on reasoning-specific data, and neither the long-context continual pretraining corpus nor the SFT data contains reasoning traces. As a result, the 8B model does not exhibit reasoning behavior by default. To elicit such behavior, we use a two-shot prompt that demonstrates step-by-step reasoning via in-context learning. The prompt is approximately 22K tokens long, and we allow the model to generate up to 8K tokens. The full prompt is shown in Listing 1. We conduct inference with a temperature of 1.0. Since the Gaokao-Phy dataset is relatively small and exhibits high variance, we repeat the experiments four times and report the average score. We use `gpt-4o-2024-11-20` (Hurst et al., 2024) as our LLM judge and run the prompt shown in Listing 2. We use greedy

generation for the LLM judger. We do not evaluate the 1B and 3B models as their reasoning performance is weaker.

Listing 1. Test prompt used in reasoning tasks. Blue-highlighted text denotes comments and is excluded from the actual model input.

```
Question: ## Task B-1.3. [The first example]

A ship traveling along a river has covered $24 \mathrm{~km}$ upstream and $28 \mathrm{~km}
$ downstream. For this journey, it took half an hour less than for traveling $30 \mathrm{~
km}$ upstream and $21 \mathrm{~km}$ downstream, or half an hour more than for traveling
$15 \mathrm{~km}$ upstream and $42 \mathrm{~km}$ downstream, assuming that both the ship
and the river move uniformly.

Determine the speed of the ship in still water and the speed of the river.



Answer: <think>
Okay, so I need to find the speed of the ship in still water and the speed of the river.
...
</think>
Let \\( v \\) be the speed of the ship in still water (in km/h) and \\( r \\) be the speed
...
The speed of the ship in still water is \\(\\boxed{10}\\) km/h and the speed of the river



Question: Prove that number $1$ can be represented as a sum of a finite number $n$ of real
 numbers, less than $1,$ not necessarily distinct, which contain in their decimal
representation only the digits $0$ and/or $7.$ Which is the least possible number $n$? [
The second example]



Answer: <think>
Okay, so the problem is to prove that the number 1 can be represented as a sum of a finite
...
</think>
To prove that the number 1 can be represented as a sum of a finite number \\( n \\) of
...
Thus, the least possible number \\( n \\) is \\(\\boxed{8}\\).



Question: {Input Question} [Real input question]
```

Listing 2. LLM-judge prompt.

```
You are a judge. Determine whether the generation explicitly contains the answer.
Only explicit mention counts -- implicit reasoning, hints, or derivations do not count.
The wording does not need to match exactly; judge based on semantic equivalence.
Ignore correctness of reasoning; only check if the answer is stated anywhere in the
generation.
If the answer appears, even if surrounded by extra text, output "yes".
Otherwise output "no".
Output only "yes" or "no".

Generation: {generation}
Answer: {answer}
```

Effiency Benchmark. All results in Table 2 are evaluated on the 8B model. We introduce *equivalent batch size* (EB) to control the on-GPU KV cache size. For a method x with per-batch on-GPU KV cache size M_x, its equivalent batch size under a real batch size B is defined as

$$EB_x = \frac{BM_x}{M_{\text{NOSA}}}. \tag{7}$$

Conversely, given a target equivalent batch size EB, the corresponding real batch size is

$$B_x = EB \cdot \frac{M_{\text{NOSA}}}{M_x}. \tag{8}$$

For example, if x is FULLATTN, at a 16K input length, NOSA retains only 4K tokens on GPU, yielding $M_{\text{NOSA}}/M_x = 0.25$. Thus, for FULLATTN, $B_x = EB/4$, whereas for NOSA, $B_{\text{NOSA}} = EB$. By controlling EB, we ensure matched on-GPU KV cache sizes across methods for fair comparison. For each setting, we report the decoding throughput averaged acrossed 4 runs after 1 warmup run, where the time is measured for decoding 4 tokens at each run. We assume that each baseline has been optimized to its fullest extent by the original authors, as efficiency is also a key concern in the corresponding works. Moreover, it would be difficult, and in some cases impossible, to implement and compare all methods within a unified system, since system designs in the KV cache offloading literature are often tailored to specific algorithmic choices.

C. Baselines

In this section, we analyze how NOSA differs from reference and baseline methods, and briefly discuss other potentially related methods.

C.1. Reference and Baseline Methods

INFLLMV2. INFLLMV2 is a trainable sparse attention mechanism closely related to NSA (Yuan et al., 2025), MOBA (Lu et al., 2025), and SEERATTENTION (Gao et al., 2025). Compared with these methods, it features a simpler design and supports switching between dense and sparse inference modes; its inference code and model checkpoints have also been released. INFLLMV2 adopts block-wise sparsity patterns without an explicit locality constraint. That is, when deployed in offloading systems, each query token may fetch an arbitrary number of blocks from CPU to GPU. NOSA improves upon this design by introducing a locality constraint and partitioning the selected blocks into query-aware and query-agnostic components. INFLLMV2 further argues that sparse attention should be incorporated into training starting from the long-context continual pretraining stage. We follow this training setting in NOSA, and show that NOSA achieves notably higher decoding efficiency with no substantial performance degradation compared with INFLLMV2.

DMA. DMA is a sparse attention mechanism that uses an importance score computed from each token's key vector to determine which tokens to attend to. It is an element-wise method: instead of selecting blocks, it directly selects individual tokens based on their importance scores. However, because this selection is query-agnostic, it effectively implements an eviction policy, i.e., once a token is discarded, it cannot be recalled. NOSA incorporates query-agnostic selection and implements the eviction head using DMA, since some KV pairs are broadly useful across queries and therefore do not require query-dependent selection. We note that the original DMA paper also proposes a query-aware variant, which corresponds to DMA-R in our experiments. In practice, DMA is ill-suited for recall-heavy tasks, and DMA-R may underperform due to training–inference mismatch. NOSA addresses these limitations by combining query-aware and query-agnostic selection, and by integrating an offloading system that balances performance and efficiency.

SHADOWKV. SHADOWKV is a training-free KV cache offloading system that reduces PCIe communication overhead by applying SVD to the K cache and offloading only the V cache. During prefill, SHADOWKV uses full attention and constructs both block representations and a low-rank representation of K. At each decoding step, it first performs query-aware block selection, and then overlaps K reconstruction with fetching V blocks from CPU. Because sparsity is introduced only during decoding, SHADOWKV suffers from a non-negligible training–inference mismatch, which can introduce approximation errors in decoding. These errors may accumulate over long generations and lead to noticeable degradation in generation quality. In addition, the overlapped K reconstruction and V fetching is not sufficiently efficient, as it relies on the model's inherent locality, which is substantially weaker than the constrained locality enforced by NOSA.

ARKVALE. The main contribution of ARKVALE is its query-aware selection strategy based on a bounding cuboid over the key vectors within each block. This selection is accurate and expressive. However, due to its complexity, this mechanism is difficult to integrate into model training and is therefore used only at inference time in the official ARKVALE setting. Consequently, ARKVALE suffers from a training–inference mismatch, which can hurt long-context performance. Moreover, due to implementation overheads, ARKVALE is notably slower than other baselines in our experiments and therefore does not improve decoding efficiency.

INFLLM. INFLLM was originally designed to support inference over longer sequences under limited GPU memory. To this end, it combines sparse attention with chunked prefill, enabling sparsity to be applied during the prefill stage. Specifically, it

builds block-level representations from the KV cache and computes query-aware relevance scores to select the most relevant blocks for each query token. However, because the sparsity pattern is not learned during training, introducing sparsity into long-context prefill can substantially degrade recall, leading to poor performance on long-context benchmarks. Moreover, the INFLLM system is not well optimized for large batch sizes and therefore fails to achieve high decoding throughput when the batch size increases.

C.2. Other Potentially Related Methods

Since NOSA is positioned as a KV cache offloading method, we briefly discuss other KV-cache optimizations, which fall into three categories: KV cache compression, quantization, and eviction.

KV Cache Compression. KV cache compression can also enable larger decoding batch sizes by reducing KV cache memory footprint. Existing KV-cache compression techniques can be broadly categorized into four levels: token-level, head-level, layer-level, and dimension-level. Token-level compression methods, such as CLUSTERATTN (Zhang et al., 2025e) and SQUEEZEDATTN (Hooper et al., 2025), merge adjacent or semantically similar KV units along the sequence length dimension. These approaches can be combined with block-wise sparse attention to enable more accurate KV selection, as in CLUSTERKV (Liu et al., 2025). Head-level compression alleviates GPU memory pressure by pruning attention heads. For example, DUOATTN (Xiao et al., 2025) and PRULONG (Bhaskar et al., 2025) propose post-training procedures to identify heads whose KV states can be removed and prune them accordingly. Layer-level methods, such as LCKV (Wu & Tu, 2024) and MINICACHE (Liu et al., 2024b), target redundancy across layers by sharing a single set of KV states among multiple layers. Dimension-level compression reduces the KV representation size via low-rank factorization (Saxena et al., 2024; Sun et al., 2025) or by performing attention computation in a latent space (Liu et al., 2024a; Mu et al., 2025; Singhania et al., 2024), thereby lowering peak GPU memory usage. Most of the above approaches, except for trainable latent-space attention such as MLA (Liu et al., 2024a), introduce a training–inference mismatch, which can substantially degrade long-form generation quality. **These techniques are orthogonal to NOSA, since they mainly target the KV cache, that is not modified by NOSA.** In addition, MLA typically requires substantially more training effort (e.g., training from scratch or an additional conversion stage to transform GQA models into MLA models (Ji et al., 2025)) than NOSA. We leave combining NOSA with other KV-cache compression techniques for future work. KV cache compression is also widely used in more complex application settings, including retrieval-augmented generation (Cao et al., 2025), code generation (Zhang et al., 2025g), and multimodal models (Jiang et al., 2025). Task semantics have also been leveraged for KV cache optimization (He et al., 2025). **NOSA can be applied to these scenarios and tasks as well, since they do not directly conflict with NOSA's design principles.**

KV Cache Quantization. KV cache quantization is a mainstream approach to reducing the KV cache memory footprint, and it can additionally benefit from low-bit kernels to accelerate inference. KVQUANT (Hooper et al., 2024) and KIVI (Liu et al., 2024c) quantize the KV cache to 4 bits and 2 bits, respectively, achieving substantial compression compared with the standard half-precision floating-point representation. More recent work has further pushed the limit toward 1-bit KV cache quantization (Zandieh et al., 2025; Tao et al., 2025). These methods are potentially compatible with quantized attention techniques, such as SAGEATTENTION (Zhang et al., 2025c;a;d;b). **NOSA is also orthogonal to KV cache quantization**, and we leave a systematic study of their combination to future work.

KV Cache Eviction. KV cache eviction methods reduce the memory burden by discarding less important KV units. Early approaches such as STREAMINGLLM (Xiao et al., 2024b) and LM-INFINITE (Han et al., 2024a) rely on fixed eviction patterns to remove cache entries. Classical methods, including SNAPKV (Li et al., 2024), H2O (Zhang et al., 2023b), and FASTGEN (Ge et al., 2024), introduce more sophisticated eviction policies, including query-aware selection that leverages the current query as prior information to identify important KV entries. Query-agnostic eviction methods, such as INFINIPOT (Kim et al., 2024) and LOCRET (Huang et al., 2025), further enable eviction in settings where the query is not immediately available. In general, these methods suffers from irreversible information loss: once a token is evicted, it cannot be retrieved. **Therefore, KV cache eviction is expected to underperform KV cache offloading. However, these methods provide a strong query-agnostic importance estimator, and they can be integrated as the eviction head in NOSA.**

D. System Optimizations in NOSI

Kernel Fusion. We utilize FLASHINFER's (Ye et al., 2025) implementations for LayerNorm, RoPE, and FFN. Since the eviction heads have fewer parameters, running them alone incurs substantial overhead; therefore, we fuse the eviction heads with splitting QKV tensor into a single kernel. As the max-pooling stage is applied identically to both query-aware and

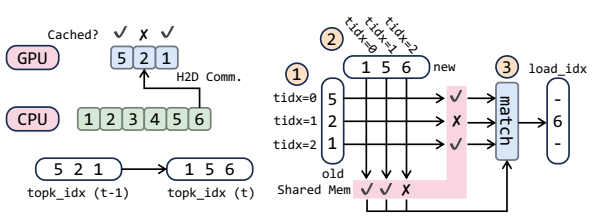


Figure 4. Memory layout and block selection.

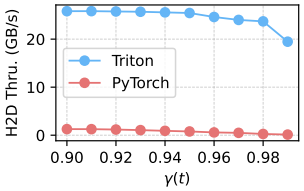


Figure 5. Throughput (↑) of host-to-device communication.

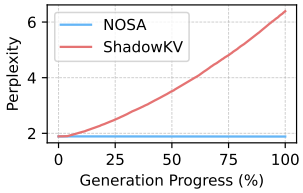


Figure 6. Case study: perplexity (↓) in long-generation.

query-agnostic scores, we also fuse the two poolings into one kernel. Both fused kernels are hand-crafted in `Triton`. We further use `CUDA Graphs` to reduce the kernel launch overhead incurred by the process from max-pooling to generating the selected indexes.

Memory Layout and Block Selection. We organize the KV cache into minimal block-sized units on both GPU and CPU (Figure 4) to reduce communication overhead. Since decoding does not require KV blocks on GPU to be contiguous, we only swap out blocks not used in the current step and replace them with the newly selected ones. This requires solving a set-difference problem and producing a replacement mapping. Since implementing this efficiently in `Pytorch` is hard, we use a custom `CUDA` kernel (Figure 4). For each thread, the kernel runs in $O(k)$, as it checks whether each old index appears in the new set, then uses shared memory to test whether new indices already exist in the old set, and finally performs a per-thread sequential pass to generate the final mapping, which is parallel-friendly.

Offloading Communication. Since NOSA selects blocks independently for each attention head, each block is relatively small (16 KiB in our setting). This fine-grained partitioning results in fragmented host-to-device communication and poor PCIe bandwidth utilization. To mitigate this issue, inspired by Zhou et al. (2025), we implement a custom `Triton` kernel for host-to-device transfers and directly leverage Unified Virtual Addressing (UVA) to access CPU memory. To evaluate its effectiveness, we compare our kernel with the vanilla `PyTorch` implementation using same experimental settings as Figure 8a. Figure 5 shows that our method outperforms the `PyTorch` baseline, reaching up to 83% of peak PCIe bandwidth, while `PyTorch` implementation achieves < 2 GB/s throughput.

E. Supplementary Empirical Results

E.1. General Short-Context Tasks and Reasoning Tasks

Table 4. The performance (%) on ***general tasks***[3]. All models are evaluated with full attention inference.

Model	Method	MM	MMP	BB	GS	MA	DR	MB	HE	Avg. ↑
1B	FullAttn	50.3	24.2	37.8	48.5	12.0	8.6	42.4	43.3	33.4
	InfLLMv2	48.5	24.2	37.6	50.9	11.7	8.6	40.0	39.0	32.6
	DMA	48.8	23.8	37.8	51.9	11.6	7.2	40.0	43.3	33.0
	NOSA	48.8	23.8	39.1	51.0	11.6	7.2	40.2	45.7	33.4
3B	FullAttn	60.1	34.8	54.2	68.5	17.4	8.6	51.6	56.7	44.0
	InfLLMv2	60.4	35.2	51.4	68.8	17.7	8.6	51.6	55.5	43.6
	DMA	60.0	35.8	52.8	67.3	17.2	8.3	53.2	54.3	43.7
	NOSA	59.9	34.8	53.4	68.2	17.5	8.5	51.8	57.3	43.9
8B	FullAttn	73.6	46.1	61.9	76.3	22.2	11.3	62.6	69.5	52.9
	InfLLMv2	73.2	45.6	63.1	77.1	22.2	10.3	63.0	67.7	52.8
	DMA	69.8	45.5	69.4	76.7	22.3	11.1	62.2	69.5	53.3
	NOSA	69.8	45.8	68.9	77.8	22.3	10.6	62.6	71.3	53.6

Table 5. The performance (%) on the ***reasoning tasks***. All experiments are conducted on the 8B model.

Dataset	FullAttn	InfLLMv2	ShadKV	InfLLM	ArkVale	DMA	**NOSA**
Math-500	52.8	50.6	19.0	38.8	47.0	**50.4**	**50.4**
Gaokao-MS	55.6	49.5	7.0	36.9	43.9	**57.0**	51.9
Gaokao-MH	61.5	57.8	15.6	42.2	47.7	58.7	**62.4**
Gaokao-Phy	36.3	34.8	8.2	29.7	30.5	31.6	**33.6**
Avg. ↑	51.6	48.2	12.5	36.9	42.3	49.4	**49.6**

We conduct experiments on general tasks (Table 4) and long-generation reasoning tasks (Table 5).

NOSA does not harm short-context general ability. As shown in Table 4, all methods achieve comparable performance

[3]See Appendix B.3 for *general* task abbreviations.

on the general tasks across model sizes. Both NOSA and DMA exhibit no significant performance degradation compared with INFLLMV2 and FULLATTN.

Training-free offloading methods fail on long-generation reasoning tasks. Table 5 shows that training-free offloading methods incur substantial performance drops on reasoning tasks. In contrast, DMA and NOSA perform better, as their sparsity patterns are learned during training. To provide more insight, we conduct a case study and measure perplexity over one generation in Figures 6. Training-free offloading (SHADOWKV) shows sharply increasing perplexity, whereas NOSA's perplexity remains low, indicating that training-inference mismatch accumulates error and corrupts reasoning generations (as shown in Appendix E.4).

E.2. Long-Context Performance

We report the full results for all three models (1B, 3B, and 8B) in Table 11.

NOSA achieves the best overall performance among all competing methods. Across the 12 settings in Table 11, NOSA attains the highest score in 9 settings, while SHADOWKV and ARKVALE perform best in 2 and 1 settings, respectively. Overall, NOSA outperforms the other methods in most cases and achieves the largest number of wins.

NOSA benefits from larger models. Sparse attention methods tend to be less stable on smaller models, which are generally less robust than larger ones. For example, on the 1B model, applying INFLLMV2 alone leads to a recall drop of more than 20% on HELMET, and all sparse attention methods exhibit notable performance degradation on HELMET at this scale. In contrast, for larger models (8B), NOSA consistently achieves the highest scores among all competing methods, and the degradation in recall performance is substantially less severe than that observed on smaller models. These results indicate that sparse attention methods, particularly NOSA, benefit from increased model capacity and larger-scale training data, which together yield a more robust base model.

E.3. Generalization to Longer Contexts

To compare NOSA with other baselines on longer input lengths and to test its ability to generalize to out-of-domain (OOD) context lengths, we evaluate all methods using the 8B model on HELMET with 32K and 64K inputs. Note that our models are trained with a 16K context window, so these evaluations correspond to $2\times$ and $4\times$ length generalization beyond the training length. For the 32K setting, we use the same protocol as in Section 4.2. For 64K, we increase RoPE's θ from 10000 to 40000 to improve performance, and increase the sparsity budget (i.e., k) to 6144. The results are reported in Table 10, from which we draw the following conclusion.

NOSA also shows superior performance in length extrapolation. As shown in Table 10, NOSA outperforms all other baselines at both 32K and 64K input lengths. Even when evaluated on OOD context lengths, NOSA maintains strong performance compared to alternative methods. These results further demonstrate that NOSA is compatible with length extrapolation, as increasing RoPE's θ is a common practice for extending the effective context length without retraining, and NOSA still performs better than the competing approaches under this modification.

E.4. Case Study

We provide a case study in Figure 6 and Figure 7. As generation length increases, training-free offloading methods (e.g., SHADOWKV) can suffer from a training-inference mismatch, leading to accumulated approximation errors over generation. As a result, the generated text becomes corrupted, as shown in Figure 7, whereas NOSA maintains generation quality.

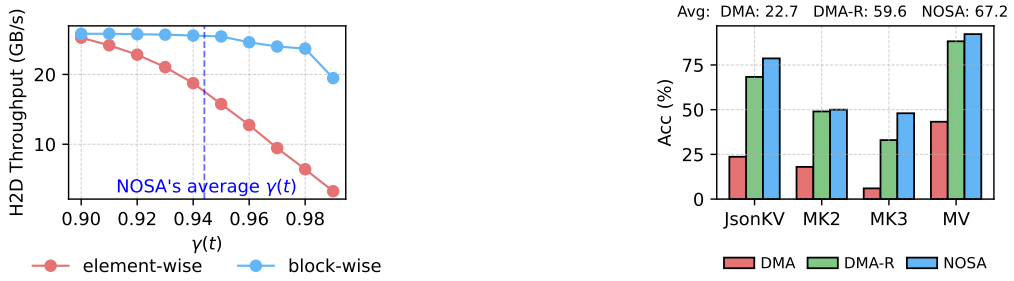
To verify this quantitatively, we sample two generations from SHADOWKV and NOSA, and report perplexity throughout the generation process. Perplexity is computed on prefixes from the input up to each position. For a fair comparison, we use the FULLATTN 8B model as the evaluator to score both generations. As shown in Figure 6, SHADOWKV exhibits a sharp increase (i.e., an explosion) in perplexity, while NOSA's perplexity remains consistently low. Higher perplexity indicates increased uncertainty in next-token prediction, which typically leads to more random generations and degraded text quality, matching the observed failure cases of SHADOWKV. In contrast, this behavior is not observed for NOSA, suggesting that maintaining consistency

Training-free Offloading (ShadowKV)	NOSA
Step 1: To start, we need to understand the relationship between the number of dogs each person is responsible for and their <skip lines> \r\n Old\r\n \r\n Until Next\r\n \r\n \r\n \r\n \r\n /\r\n	Okay, let's break down the problem step by step. First, we know Denali is responsible for 16 dogs and Nate for 12 dogs. <skip lines> The value of \(x\) is \(\boxed{2}\). </think>

Figure 7. Case study of a single generation on MATH-500.

Table 6. Performance of various eviction head implementations.

Implementation	INFLLMV2	LOCRET	DMA	S-DMA	**ED-DMA**
RULER-16K (%) ↑	60.3	55.9	56.2	60.9	**61.9**



(a) Block-wise vs. element-wise host-to-device throughput *(b)* DMA, DMA-R, and NOSA's accuracies on HELMET's recall.

Figure 8. Comparison between DMA and NOSA.

between training and inference prevents error accumulation and enables consistently high-quality long-generation.

E.5. Ablation Studies

E.5.1. Ablation on Algorithm Design

We ablate the core components of NOSA in this section.

Eviction Head Implementation. To demonstrate that ED-DMA is the most suitable implementation for the eviction head, we compare ED-DMA with several alternative designs, including LOCRET (Huang et al., 2025), vanilla DMA (Shi et al., 2025), and simple-DMA. Specifically, LOCRET employs a lightweight trainable MLP to produce the importance score, i.e., $s_j^e = \mathrm{MLP}(\mathbf{h}_j)$. Vanilla DMA selects important blocks based on $\exp(b_j)$ without delaying the $\exp$ operation. Simple-DMA (S-DMA) applies a straight-through estimator to the attention bias b_j in ED-DMA; as a result, its attention computation is identical to vanilla attention. More details on each implementation and the full results are included in Appendix E.5.2.

We apply these implementations and conduct long-context continual pretraining on a 1B-size model, then evaluate on RULER (Hsieh et al., 2024) with 16K input lengths. As shown in Table 6, ED-DMA outperforms all other implementations. Notably, the large performance degradation of vanilla DMA indicates the significant negative impact of the precision loss introduced by the $\exp$ operation.

Locality. We compare the locality $\gamma(t)$ for each transformer layer of the trained 1B-size model using an input sequence of 16K tokens sampled from PG19 (Rae et al., 2020). As shown in Figure 2d, NOSA obtains a higher locality compared with the original INFLLMV2, yielding an approximate 5.5% improvement (equivalent to nearly a $2\times$ reduction in cache miss rate in KV cache offloading systems). These results demonstrate the effectiveness of incorporating a locality constraint to enhance locality.

Comparison with DMA. Since the eviction head in NOSA is inspired by DMA, we provide an intuitive discussion of how our design differs from DMA. Specifically, (1) DMA performs element-wise selection, whereas NOSA selects KV cache in a block-wise manner; and (2) NOSA uses query-aware selection with query-agnostic selection, while DMA only relies on query-agnostic selection.

Element-wise selection is ill-suited for KV cache offloading. The large number of irregular offsets and highly fragmented host-to-device (CPU to GPU) transfers result in poor PCIe bandwidth utilization. We compare the two communication patterns under batch size $B = 64$, sequence length $n = 4096$, head dimension $d_{\text{kv}} = 128$, and $n_{\text{kv}} = 2$ KV heads, across different locality levels $\gamma(t)$. We only fetch KV units that are not resident on the GPU, i.e., a fraction of $1 - \gamma(t)$. Both kernels are implemented in `Triton` and use UVA to directly access host memory. Figure 8a shows that element-wise communication degrades sharply under high locality, whereas block-wise communication sustains high throughput, motivating our use of the block-wise pattern.

We introduce query-aware selection in NOSA because DMA's query-agnostic policy is inadequate for long-context recall, as it causes irreversible eviction. We evaluate DMA and NOSA on HELMET recall tasks (Yen et al., 2025) (MK2, MK3, MV from RULER (Hsieh et al., 2024), and JsonKV), and also include DMA-R (trained with DMA, inference with NOSA).

Figure 8b shows DMA has low recall. Query-aware selection at test time improves significantly, and NOSA further outperforms DMA-R, suggesting query-aware selection is necessary in both training and inference.

E.5.2. Eviction Head Implementation

Here, we ablate the implementation of the eviction head to evaluate the effectiveness of NOSA's design. The eviction head assigns an importance score s_j^e to each token at position j, which serves as the metric for selecting query-agnostic tokens in NOSA. There are several possible approaches to implement this mechanism. A straightforward approach is to learn the importance score through a lightweight MLP that takes the hidden state of each token as input, similar to the *retaining head* proposed in LOCRET (Huang et al., 2025). Alternatively, DMA (Shi et al., 2025) employs a single-layer gated projection based on $\mathbf{v}_j$, the value vector at the j-th token, to estimate importance. In NOSA, we perform query-agnostic selection using the pre-exponential value from DMA, motivated by our observation that this step is highly sensitive to numerical precision. This variant is referred to as Exp-Delayed DMA (ED-DMA). For comparison, we also include a variant where no explicit attention bias is applied during forward passes, and only its gradient contribution is preserved during backpropagation. This simplified version is termed Simple DMA (S-DMA). We list the bias calculation before query-agnostic selection and the attention calculation in Table 7.

We first pretrain a 1B-parameter model with an input sequence length of 8K under dense attention, and then perform long-context continual pretraining with NOSA using different eviction head implementations, where the sequence length is extended to 16K. We set the total selection budget k to 4096, with 64 attention sink tokens, 1024 sliding window tokens, and $k_q = 1024$. The resulting models are evaluated on RULER (Hsieh et al., 2024). As shown in Table 8, ED-DMA achieves the best performance, remaining nearly lossless compared to the original INFLLMV2. Other implementations exhibit noticeable performance degradation, particularly on the MK and MV tasks. Notably, merely shifting the exponential operator from the bias calculation to the attention computation yields a 5.7% improvement in accuracy. Based on these results, we adopt ED-DMA as the eviction head implementation for NOSA.

Table 7. Details of various implementations of the eviction head.

Abbreviation	Full Name	Bias Calculation	Attention Calculation
LOCRET	LOCRET	$b_j = \sigma(\mathbf{h}_j\mathbf{W}_1)\mathbf{W}_2$	$\mathbf{o}_i = \sum_j \frac{\exp(\mathbf{q}_i\mathbf{k}_j^\top + b_j + m_{ij})\mathbf{v}_j}{\sum_l \exp(\mathbf{q}_i\mathbf{k}_l^\top + b_l + m_{il})}$
DMA	Dynamic Mask Attention	$b_j = \exp(\tau(\mathbf{v_j}\mathbf{W_1}) \odot \mathbf{W_2})$	$\mathbf{o}_i = \sum_j \frac{b_j \exp(\mathbf{q}_i\mathbf{k}_j^\top + m_{ij})\mathbf{v}_j}{\sum_l b_l \exp(\mathbf{q}_i\mathbf{k}_l^\top + m_{il})}$
ED-DMA	Exp-Delayed DMA	$b_j = \tau(\mathbf{v_j}\mathbf{W_1}) \odot \mathbf{W_2}$	$\mathbf{o}_i = \sum_j \frac{\exp(b_j) \exp(\mathbf{q}_i\mathbf{k}_j^\top + m_{ij})\mathbf{v}_j}{\sum_l \exp(b_l) \exp(\mathbf{q}_i\mathbf{k}_l^\top + m_{il})}$
S-DMA	Simple-DMA	$b_j = \tau(\mathbf{v_j}\mathbf{W_1}) \odot \mathbf{W_2}$	$\mathbf{o}_i = \sum_j \frac{\exp(b_j - b_j.\text{detach}()) \exp(\mathbf{q}_i\mathbf{k}_j^\top + m_{ij})\mathbf{v}_j}{\sum_l \exp(b_l - b_l.\text{detach}()) \exp(\mathbf{q}_i\mathbf{k}_l^\top + m_{il})}$

Table 8. RULER evaluation results with various implementations of the eviction head.

Implementation	SG1	SG2	SG3	MK1	MK2	MK3	MV	MQ	VT	CWE	FWE	QA1	QA2	Avg. ↑
INFLLM-V2	100.0	100.0	100.0	84.0	50.0	18.0	92.5	84.5	26.0	0.8	62.7	36.0	30.0	60.3
LOCRET	**100.0**	98.0	86.0	64.0	40.0	4.0	77.0	**79.5**	41.6	3.6	64.7	36.0	32.0	55.9
DMA	**100.0**	**100.0**	**98.0**	70.0	**44.0**	8.0	75.5	70.0	20.8	2.0	74.7	36.0	32.0	56.2
S-DMA	**100.0**	96.0	**98.0**	60.0	42.0	8.0	87.0	77.0	**94.8**	3.6	60.7	30.0	**34.0**	60.9
ED-DMA	**100.0**	98.0	**98.0**	**76.0**	42.0	**14.0**	**90.5**	78.5	62.4	**3.8**	**68.0**	**40.0**	**34.0**	**61.9**

E.5.3. Hyperparameter Analysis

Varying the ratio of query-aware selection k_q/k. The minimum locality, $\gamma_0 = k_e/k = 1 - k_q/k$, is a crucial hyperparameter that controls the number of blocks selected during query-aware selection, thereby affecting both the performance and decoding efficiency of NOSA. We evaluate NOSA on the 8B size model on HELMET, following Appendix B's setting, where we set $k = 4096$ and $k_q \in \{512, 1024, 1536, 2048\}$ at inference time. We sample 20 entries for each task and report the averaged score, followed by the decoding throughput under the $EB = 16$ and 16K input-length setting,

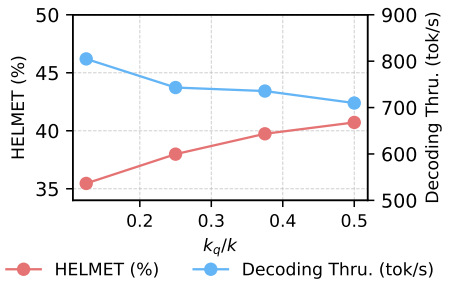


Figure 9. HELMET score and decoding throughput under various k_q/k.

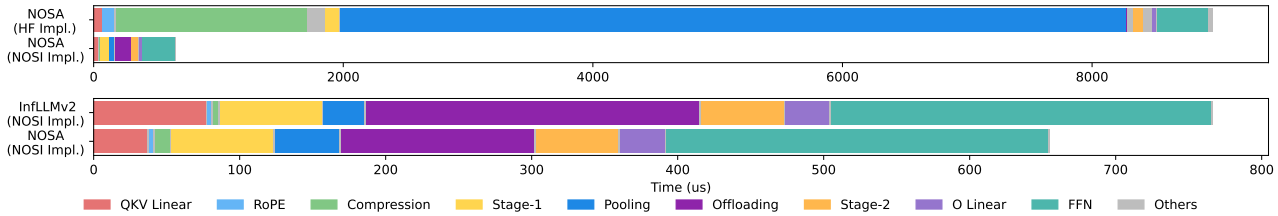


Figure 10. Decoding wall-time breakdown analysis.

in Figure 9.

Changing k_q/k trades long-context performance for decoding efficiency. Figure 9 shows a clear performance–throughput trade-off as k_q/k varies. Decreasing k_q/k enforces higher locality, which increases decoding throughput but weakens query-aware selection, limiting recall and reducing task performance. Conversely, increasing k_q/k improves performance at the cost of lower throughput. This tunability allows adjusting k_q/k at inference to meet different requirements: use larger k_q/k when quality is prioritized, and smaller k_q/k when higher throughput is needed. Moreover, although the model is trained with k_q/k, it generalizes well to other ratios at inference time, demonstrating the robustness and generalizability of the hyperparameter k_q/k. To balance long-context performance and efficiency, we set $k_q/k = 0.25$ in our main experiments, under which NOSA outperforms the baselines in both performance and efficiency.

Varying the budget k. To demonstrate that NOSA consistently achieves better performance across different levels of attention sparsity (i.e., different budget k), we conduct an ablation study on k. We benchmark NOSA and all baselines on HELMET using the 8B model, setting $k \in 2048, 4096, 6144$ to vary the sparsity level. We set the sliding window size to 1024 for $k \in 4096, 6144$ and to 512 for $k = 2048$. For NOSA, we fix the ratio $k_q/k = 0.25$ across all settings. Other settings are aligned to Appendix B. For each task, we sample 20 instances and report the results in Table 9.

A larger budget improves long-context performance. As shown in Table 9, the average HELMET score increases as the budget k grows. When the sparsity reaches 0.375, NOSA shows no performance degradation under either full prefill or sparse prefill compared with FULLATTN. In contrast, with smaller budgets, long-context performance decreases accordingly. The results also indicate that Recall and Rerank are more sensitive to changes in k, whereas other tasks, especially ICL and RAG, remain relatively stable across different budgets.

NOSA outperforms all baselines across various budgets. NOSA achieves a higher average HELMET score than all other baselines under both full-prefill and sparse-prefill settings. As the budget decreases, NOSA maintains strong performance, whereas the baselines exhibit notable performance degradation. For more complex long-context tasks, such as Cite and Rerank, the baselines suffer a larger degree of degradation compared with NOSA. These results demonstrate that NOSA is more robust to varying levels of attention sparsity.

E.5.4. WALL-TIME BREAKDOWN ANALYSIS

To demonstrate the necessity of constructing the NOSI framework rather than evaluating efficiency within the Hugging Face (HF) implementation, and to highlight the impact of reduced KV-loading time, we conduct a wall-time breakdown analysis on the 8B model with a 16K input length and an equivalent batch size of 16. We compare NOSA with NOSI implementations against the Hugging Face implementation, and additionally include a comparison with INFLLMV2 implemented in NOSI. The results are shown in Figure 10. We decompose the decoding time of a single Transformer layer into nine components, and group all remaining overheads (e.g., kernel launches, tensor reshaping, and CPU-side processing) under "Others".

The vanilla Hugging Face implementation is highly inefficient and fails to expose the KV-loading overhead. As shown in the upper half of Figure 10, the HF implementation of NOSA incurs extremely large latencies in block compression and pooling, mainly due to excessive `PyTorch` kernel invocations and inefficient execution. Under this implementation, the KV-loading time in NOSA (the purple segment in the second line) is negligible compared with the time spent on compression and pooling. Consequently, optimizing KV-loading yields little end-to-end benefit when using the HF implementation. In contrast, NOSI provides a much more efficient implementation, which exposes KV-loading as a non-trivial component of the decoding latency. Therefore, NOSI enables a meaningful evaluation of KV-loading optimizations and can demonstrate their practical impact on end-to-end performance.

NOSA achieves a significant reduction in KV-loading time compared with INFLLMV2. By explicitly enforcing a locality constraint, NOSA loads fewer blocks from CPU at each decoding step, resulting in substantially lower KV-loading latency than INFLLMV2, as shown in the lower half of Figure 10. In particular, the KV-loading time is reduced by about half. In addition, NOSA reduces the QKV linear projection time by using a fused kernel that splits the QKV tensor while simultaneously computing the importance scores. In contrast, INFLLMV2 does not compute importance scores and relies on `PyTorch`'s `split` operator, which incurs higher overhead than our fused kernel.

F. Theoretical Guarantee of Locality

Theorem 3.1. *If the selection process of NOSA has budget $k = k_q + k_e$, we have $\forall t \in \{2, \cdots, n\},\ \gamma(t) \geq \frac{k_e}{k}$.*

Proof. Such statement equals to $|\Gamma(t-1) \cap \Gamma(t)| \geq k_e$. We only prove the token-wise selection case below for brevity, since the block-wise selection case follows the same reasoning. For simplicity, we do not consider the newly generated token at the t-th step, as it always resides on the GPU. Accordingly, we assume that k tokens are selected from n tokens at both the $(t-1)$-th and t-th steps.

Denote the block indexes of query-aware selection at step $t-1$ as

$$\Gamma_q(t-1) = \{a_1, a_2, \cdots, a_{k_q}\}, \text{ such that } s^q_{a_1} \geq s^q_{a_2} \geq \cdots \geq s^q_{a_{k_q}}. \tag{9}$$

Denote the indexes of the Top-k largest importance scores as

$$\text{ArgTop}_k(\{s^e_i\}_{i=1}^n) = \{b_1, b_2, \cdots, b_k\}, \text{ such that } s^e_{b_1} \geq s^e_{b_2} \geq \cdots \geq s^e_{b_k}. \tag{10}$$

Here, $\{b_1, \cdots, b_k; a_1, \cdots, a_{k_q}\}$ is a multiset, since some indices in $\{b_1, \cdots, b_k\}$ may also appear in $\{a_1, \cdots, a_{k_q}\}$. According to NOSA's selection process, we have $\Gamma(t-1) \subseteq \{b_1, \cdots, b_k; a_1, \cdots, a_{k_q}\}$. Denote $p = |\{b_1, \cdots, b_k; a_1, \cdots, a_{k_q}\}| - |\Gamma(t-1)|$, then $1 \leq p \leq k_q$.

Now, consider the elements in $\{b_1, \cdots, b_k; a_1, \cdots, a_{k_q}\}$ that are not contained in $\Gamma(t-1)$. Since query-aware selection is prioritized, we have $\{a_1, \cdots, a_{k_q}\} \subset \Gamma(t-1)$. Due to the monotonic construction, the last p elements in $\{b_1, \cdots, b_k\}$ are not included in $\Gamma(t-1)$.

Therefore, we have

$$b_1, b_2, \cdots, b_{k-p} \in \Gamma(t-1). \tag{11}$$

Since $0 \leq p \leq k_q$, we must have $b_1, b_2, \cdots, b_{k_e} \in \Gamma(t-1)$. Note that all the above derivations are independent of t. Similarly, we have $b_1, b_2, \cdots, b_{k_e} \in \Gamma(t)$, thereby implying $|\Gamma(t-1) \cap \Gamma(t)| \geq k_e$.

□

Table 9. ***Helmet*** scores (%) of NOSA compared with all baselines under various budget k. NOSA^F uses full attention during the prefill stage and sparse attention during the decode stage, while NOSA^S adopts sparse attention for both stages.

Tasks	*Full Prefill*						*Sparse Prefill*						
	FullAttn	ShadKV$_{64}$	ShadKV$_8$	ArkVale	DMA^F	**NOSA^F**	InfLLMv2	ShadKV$^M_{64}$	ShadKV$^M_8$	InfLLM$_{128}$	InfLLM$_{64}$	DMA^S	**NOSA^S**
Budget $k = 2048$ ***Sparsity: 0.125***													
Recall	88.0$_{\pm0.8}$	40.3$_{\pm4.3}$	72.6$_{\pm1.1}$	49.3$_{\pm3.8}$	17.5$_{\pm1.1}$	**73.9**$_{\pm0.8}$	54.2$_{\pm0.9}$	30.5$_{\pm3.0}$	**56.0**$_{\pm2.8}$	20.3$_{\pm2.3}$	20.4$_{\pm2.2}$	10.7$_{\pm2.1}$	40.4$_{\pm2.5}$
RAG	54.5$_{\pm1.9}$	53.3$_{\pm2.5}$	53.8$_{\pm2.2}$	54.0$_{\pm2.6}$	52.2$_{\pm3.2}$	**56.4**$_{\pm2.0}$	51.8$_{\pm2.3}$	44.5$_{\pm2.9}$	43.6$_{\pm1.4}$	46.3$_{\pm3.6}$	46.7$_{\pm2.2}$	28.7$_{\pm1.8}$	**46.8**$_{\pm1.4}$
ICL	59.7$_{\pm2.1}$	55.7$_{\pm0.5}$	57.3$_{\pm2.1}$	60.7$_{\pm2.6}$	**63.7**$_{\pm2.5}$	59.0$_{\pm2.4}$	66.3$_{\pm3.3}$	52.3$_{\pm2.6}$	49.3$_{\pm5.3}$	41.3$_{\pm3.4}$	42.7$_{\pm2.1}$	42.3$_{\pm3.9}$	**60.7**$_{\pm5.4}$
Cite	13.9$_{\pm3.7}$	9.7$_{\pm0.6}$	8.6$_{\pm0.5}$	10.9$_{\pm1.1}$	9.2$_{\pm2.8}$	**11.9**$_{\pm2.3}$	12.7$_{\pm0.9}$	4.9$_{\pm0.9}$	6.9$_{\pm1.5}$	8.3$_{\pm0.8}$	9.6$_{\pm1.2}$	7.8$_{\pm2.9}$	**12.7**$_{\pm1.0}$
Rerank	36.4$_{\pm3.1}$	13.3$_{\pm1.6}$	26.5$_{\pm2.3}$	**29.2**$_{\pm2.5}$	17.4$_{\pm2.2}$	24.7$_{\pm2.6}$	26.4$_{\pm1.7}$	6.0$_{\pm0.7}$	13.1$_{\pm1.2}$	8.1$_{\pm0.5}$	8.2$_{\pm0.9}$	8.0$_{\pm1.6}$	**22.6**$_{\pm1.8}$
LongQA	31.0$_{\pm2.2}$	27.9$_{\pm4.2}$	32.6$_{\pm4.3}$	28.8$_{\pm1.2}$	29.5$_{\pm3.4}$	**31.2**$_{\pm1.3}$	28.5$_{\pm3.6}$	25.0$_{\pm2.3}$	25.5$_{\pm2.9}$	9.2$_{\pm4.0}$	12.5$_{\pm4.2}$	**29.6**$_{\pm2.6}$	28.6$_{\pm4.6}$
Summ.	11.3$_{\pm4.8}$	**9.8**$_{\pm4.8}$	7.0$_{\pm3.2}$	7.6$_{\pm2.0}$	4.2$_{\pm0.6}$	7.4$_{\pm3.7}$	9.1$_{\pm3.2}$	4.1$_{\pm0.8}$	3.9$_{\pm2.5}$	8.2$_{\pm3.8}$	**9.1**$_{\pm4.5}$	6.7$_{\pm2.7}$	7.0$_{\pm1.3}$
Avg. ↑	42.1$_{\pm0.8}$	30.0$_{\pm1.2}$	36.9$_{\pm1.2}$	34.3$_{\pm0.9}$	27.7$_{\pm0.2}$	**37.8**$_{\pm0.4}$	35.6$_{\pm0.8}$	23.9$_{\pm0.5}$	28.3$_{\pm1.1}$	20.2$_{\pm0.7}$	21.3$_{\pm0.4}$	19.1$_{\pm1.0}$	**31.3**$_{\pm1.9}$
Budget $k = 4096$ ***Sparsity: 0.250***													
Recall	88.0$_{\pm0.8}$	56.3$_{\pm3.6}$	78.5$_{\pm1.1}$	58.6$_{\pm1.6}$	31.9$_{\pm4.9}$	**85.5**$_{\pm1.2}$	74.2$_{\pm3.6}$	55.7$_{\pm6.2}$	**70.5**$_{\pm3.0}$	26.8$_{\pm1.3}$	32.4$_{\pm2.7}$	20.9$_{\pm1.0}$	67.2$_{\pm1.3}$
RAG	54.5$_{\pm1.9}$	53.8$_{\pm1.7}$	53.1$_{\pm2.2}$	54.3$_{\pm2.1}$	53.1$_{\pm2.8}$	**56.2**$_{\pm3.0}$	53.1$_{\pm2.0}$	**53.4**$_{\pm1.8}$	51.8$_{\pm1.6}$	46.3$_{\pm2.1}$	46.4$_{\pm2.4}$	40.0$_{\pm2.5}$	52.6$_{\pm1.7}$
ICL	59.7$_{\pm2.1}$	57.3$_{\pm3.9}$	58.3$_{\pm0.5}$	61.3$_{\pm2.6}$	**67.3**$_{\pm0.9}$	61.3$_{\pm1.7}$	72.0$_{\pm0.8}$	53.7$_{\pm3.4}$	55.0$_{\pm1.4}$	39.0$_{\pm2.2}$	38.0$_{\pm3.7}$	59.3$_{\pm1.2}$	**67.0**$_{\pm2.2}$
Cite	13.9$_{\pm3.7}$	9.6$_{\pm2.6}$	8.9$_{\pm2.3}$	11.6$_{\pm1.3}$	9.9$_{\pm0.5}$	**12.4**$_{\pm0.5}$	12.8$_{\pm3.5}$	9.7$_{\pm1.2}$	7.9$_{\pm2.3}$	**12.8**$_{\pm1.4}$	11.2$_{\pm1.6}$	9.6$_{\pm1.5}$	10.4$_{\pm0.4}$
Rerank	36.4$_{\pm3.1}$	23.7$_{\pm1.7}$	28.6$_{\pm2.3}$	32.8$_{\pm2.6}$	25.8$_{\pm3.8}$	**33.4**$_{\pm0.9}$	35.5$_{\pm2.9}$	20.8$_{\pm3.5}$	25.5$_{\pm3.6}$	12.4$_{\pm0.5}$	14.1$_{\pm0.2}$	17.3$_{\pm1.8}$	**30.4**$_{\pm2.7}$
LongQA	31.0$_{\pm2.2}$	29.2$_{\pm4.4}$	**31.9**$_{\pm6.3}$	30.8$_{\pm1.9}$	30.7$_{\pm2.5}$	31.4$_{\pm1.7}$	31.4$_{\pm1.6}$	26.4$_{\pm3.4}$	**31.9**$_{\pm7.0}$	11.6$_{\pm3.5}$	11.1$_{\pm1.9}$	31.5$_{\pm4.6}$	28.5$_{\pm3.6}$
Summ.	11.3$_{\pm4.8}$	9.9$_{\pm4.2}$	**10.4**$_{\pm3.4}$	8.9$_{\pm3.5}$	8.3$_{\pm2.6}$	9.1$_{\pm2.4}$	11.2$_{\pm3.6}$	**11.1**$_{\pm3.0}$	8.7$_{\pm3.2}$	9.1$_{\pm2.5}$	10.0$_{\pm5.8}$	9.4$_{\pm4.3}$	8.9$_{\pm2.3}$
Avg. ↑	42.1$_{\pm0.8}$	34.3$_{\pm0.5}$	38.5$_{\pm1.7}$	36.9$_{\pm0.5}$	32.4$_{\pm0.6}$	**41.3**$_{\pm0.4}$	41.5$_{\pm0.8}$	33.0$_{\pm1.1}$	35.9$_{\pm0.1}$	22.6$_{\pm0.4}$	23.3$_{\pm1.6}$	26.9$_{\pm0.6}$	**37.9**$_{\pm0.8}$
Budget $k = 6144$ ***Sparsity: 0.375***													
Recall	88.0$_{\pm0.8}$	68.0$_{\pm2.0}$	79.7$_{\pm1.6}$	68.0$_{\pm3.0}$	46.2$_{\pm2.9}$	**93.6**$_{\pm2.2}$	86.7$_{\pm1.6}$	61.0$_{\pm4.9}$	71.4$_{\pm3.4}$	34.4$_{\pm1.4}$	38.1$_{\pm3.1}$	36.1$_{\pm2.1}$	**83.1**$_{\pm4.6}$
RAG	54.5$_{\pm1.9}$	54.0$_{\pm2.0}$	53.0$_{\pm2.6}$	55.1$_{\pm2.4}$	53.5$_{\pm3.0}$	**56.4**$_{\pm1.7}$	55.1$_{\pm2.0}$	53.3$_{\pm2.1}$	51.9$_{\pm1.3}$	46.3$_{\pm1.7}$	46.3$_{\pm2.9}$	47.6$_{\pm1.4}$	**53.6**$_{\pm0.9}$
ICL	59.7$_{\pm2.1}$	58.3$_{\pm2.4}$	58.7$_{\pm0.9}$	61.0$_{\pm2.2}$	**67.3**$_{\pm0.5}$	63.0$_{\pm2.2}$	68.0$_{\pm1.4}$	55.0$_{\pm2.2}$	55.3$_{\pm1.9}$	40.3$_{\pm2.1}$	41.3$_{\pm0.5}$	65.3$_{\pm0.9}$	**66.7**$_{\pm2.6}$
Cite	13.9$_{\pm3.7}$	10.6$_{\pm0.9}$	10.5$_{\pm1.2}$	10.5$_{\pm0.9}$	**10.7**$_{\pm1.3}$	**10.7**$_{\pm1.3}$	14.5$_{\pm4.4}$	8.8$_{\pm1.4}$	8.2$_{\pm1.7}$	**12.1**$_{\pm1.5}$	11.4$_{\pm1.9}$	10.6$_{\pm1.1}$	12.0$_{\pm1.5}$
Rerank	36.4$_{\pm3.1}$	25.4$_{\pm2.1}$	30.1$_{\pm2.2}$	33.5$_{\pm2.1}$	30.4$_{\pm1.6}$	**37.0**$_{\pm2.6}$	39.7$_{\pm1.0}$	25.0$_{\pm1.8}$	26.4$_{\pm4.0}$	18.2$_{\pm1.7}$	20.5$_{\pm0.3}$	26.3$_{\pm2.0}$	**36.9**$_{\pm1.3}$
LongQA	31.0$_{\pm2.2}$	29.7$_{\pm4.2}$	31.2$_{\pm6.2}$	30.5$_{\pm0.8}$	**32.5**$_{\pm3.6}$	31.8$_{\pm0.9}$	30.2$_{\pm1.8}$	26.1$_{\pm4.3}$	**32.1**$_{\pm7.7}$	11.4$_{\pm2.3}$	10.6$_{\pm1.4}$	31.1$_{\pm5.5}$	31.3$_{\pm2.4}$
Summ.	11.3$_{\pm4.8}$	9.0$_{\pm4.0}$	7.8$_{\pm2.5}$	10.0$_{\pm2.8}$	7.7$_{\pm1.0}$	**10.2**$_{\pm2.7}$	9.9$_{\pm4.7}$	9.9$_{\pm4.4}$	7.2$_{\pm2.7}$	10.0$_{\pm2.0}$	9.8$_{\pm4.9}$	10.8$_{\pm2.9}$	**11.5**$_{\pm1.9}$
Avg. ↑	42.1$_{\pm0.8}$	36.4$_{\pm0.5}$	38.7$_{\pm1.5}$	38.4$_{\pm0.8}$	35.5$_{\pm0.6}$	**43.3**$_{\pm0.5}$	43.4$_{\pm0.8}$	34.2$_{\pm0.3}$	36.1$_{\pm0.1}$	24.7$_{\pm0.4}$	25.5$_{\pm0.7}$	32.6$_{\pm1.1}$	**42.1**$_{\pm1.0}$

Table 10. ***Helmet*** scores (%) of NOSA compared with all baselines on longer input lengths. NOSA^F uses full attention during the prefill stage and sparse attention during the decode stage, while NOSA^S adopts sparse attention for both stages.

Tasks	*Full Prefill*						*Sparse Prefill*						
	FullAttn	ShadKV$_{64}$	ShadKV$_8$	ArkVale	DMA^F	**NOSA^F**	InfLLMv2	ShadKV$^M_{64}$	ShadKV$^M_8$	InfLLM$_{128}$	InfLLM$_{64}$	DMA^S	**NOSA^S**
Input Length: 32K													
Recall	56.3$_{\pm1.8}$	31.0$_{\pm0.6}$	48.8$_{\pm3.4}$	41.6$_{\pm2.5}$	10.0$_{\pm1.8}$	**56.6**$_{\pm4.2}$	45.3$_{\pm2.0}$	30.4$_{\pm2.7}$	**42.6**$_{\pm3.2}$	15.0$_{\pm3.2}$	15.6$_{\pm1.9}$	8.4$_{\pm2.7}$	35.0$_{\pm2.5}$
RAG	50.7$_{\pm0.5}$	49.1$_{\pm1.3}$	49.3$_{\pm2.0}$	50.2$_{\pm1.5}$	47.9$_{\pm2.5}$	**51.4**$_{\pm1.4}$	49.7$_{\pm2.2}$	44.5$_{\pm0.9}$	**46.4**$_{\pm1.6}$	44.9$_{\pm2.9}$	45.1$_{\pm3.1}$	32.2$_{\pm3.9}$	44.7$_{\pm3.2}$
ICL	67.3$_{\pm2.6}$	**66.7**$_{\pm5.7}$	**66.7**$_{\pm2.5}$	65.7$_{\pm4.9}$	59.3$_{\pm4.0}$	64.7$_{\pm3.8}$	69.7$_{\pm2.5}$	60.0$_{\pm1.7}$	64.0$_{\pm1.4}$	38.3$_{\pm2.6}$	38.3$_{\pm1.2}$	53.3$_{\pm3.3}$	**68.7**$_{\pm5.4}$
Cite	13.9$_{\pm3.7}$	9.9$_{\pm2.3}$	10.7$_{\pm0.6}$	9.7$_{\pm1.5}$	10.3$_{\pm0.2}$	**12.7**$_{\pm1.1}$	12.6$_{\pm3.7}$	9.8$_{\pm1.2}$	9.2$_{\pm1.9}$	13.0$_{\pm1.7}$	**10.9**$_{\pm1.4}$	9.5$_{\pm1.3}$	10.0$_{\pm1.9}$
Rerank	30.8$_{\pm0.5}$	9.1$_{\pm0.5}$	19.6$_{\pm1.9}$	19.4$_{\pm1.9}$	13.3$_{\pm1.0}$	**22.0**$_{\pm4.0}$	20.0$_{\pm3.3}$	6.6$_{\pm1.8}$	13.8$_{\pm2.2}$	4.3$_{\pm1.0}$	4.8$_{\pm1.7}$	3.2$_{\pm0.8}$	**16.4**$_{\pm1.6}$
LongQA	38.0$_{\pm7.0}$	34.7$_{\pm4.7}$	36.3$_{\pm5.6}$	32.8$_{\pm3.2}$	**34.8**$_{\pm5.7}$	34.2$_{\pm6.8}$	32.5$_{\pm4.7}$	33.2$_{\pm3.2}$	32.3$_{\pm2.6}$	10.6$_{\pm1.4}$	10.9$_{\pm3.4}$	**33.9**$_{\pm1.4}$	30.6$_{\pm6.0}$
Summ.	10.1$_{\pm2.7}$	**10.9**$_{\pm2.2}$	11.2$_{\pm3.0}$	8.2$_{\pm2.0}$	4.5$_{\pm1.2}$	10.6$_{\pm0.9}$	8.3$_{\pm2.1}$	8.0$_{\pm0.7}$	5.7$_{\pm1.3}$	9.0$_{\pm0.8}$	7.9$_{\pm2.4}$	7.3$_{\pm2.9}$	**9.6**$_{\pm3.4}$
Avg. ↑	38.2$_{\pm1.3}$	30.2$_{\pm1.4}$	34.6$_{\pm1.6}$	32.5$_{\pm1.1}$	25.7$_{\pm1.5}$	**36.0**$_{\pm1.5}$	34.0$_{\pm1.9}$	27.5$_{\pm0.7}$	30.6$_{\pm0.6}$	19.3$_{\pm1.2}$	19.1$_{\pm0.9}$	21.1$_{\pm0.6}$	**30.7**$_{\pm2.2}$
Input Length: 64K													
Recall	38.2$_{\pm2.8}$	20.5$_{\pm0.8}$	26.3$_{\pm2.6}$	23.0$_{\pm2.9}$	5.4$_{\pm0.9}$	**40.4**$_{\pm1.7}$	30.2$_{\pm2.6}$	16.6$_{\pm1.4}$	**29.0**$_{\pm4.6}$	12.2$_{\pm1.9}$	12.2$_{\pm2.0}$	6.8$_{\pm2.0}$	27.0$_{\pm1.5}$
RAG	44.9$_{\pm2.6}$	44.8$_{\pm1.9}$	44.7$_{\pm3.2}$	**45.3**$_{\pm2.2}$	40.5$_{\pm3.5}$	45.0$_{\pm3.5}$	42.4$_{\pm1.1}$	38.6$_{\pm1.4}$	38.2$_{\pm1.9}$	**42.6**$_{\pm3.6}$	41.7$_{\pm3.2}$	32.9$_{\pm5.3}$	42.5$_{\pm1.5}$
ICL	58.0$_{\pm3.7}$	56.7$_{\pm3.7}$	59.3$_{\pm2.9}$	**61.3**$_{\pm3.7}$	59.7$_{\pm1.2}$	49.3$_{\pm1.2}$	70.3$_{\pm4.9}$	59.3$_{\pm3.4}$	59.7$_{\pm2.6}$	44.0$_{\pm5.0}$	40.0$_{\pm2.2}$	58.0$_{\pm0.8}$	**71.3**$_{\pm6.1}$
Cite	12.9$_{\pm1.8}$	9.0$_{\pm1.5}$	10.0$_{\pm2.0}$	**12.1**$_{\pm0.7}$	11.0$_{\pm1.2}$	10.0$_{\pm1.3}$	8.8$_{\pm2.0}$	7.0$_{\pm1.1}$	8.3$_{\pm1.8}$	**12.1**$_{\pm1.5}$	11.5$_{\pm2.5}$	7.0$_{\pm1.6}$	7.6$_{\pm0.9}$
Rerank	11.2$_{\pm1.7}$	0.5$_{\pm0.2}$	2.8$_{\pm1.1}$	**9.5**$_{\pm1.2}$	7.4$_{\pm0.4}$	8.6$_{\pm0.7}$	6.8$_{\pm2.6}$	0.8$_{\pm0.8}$	0.8$_{\pm0.7}$	1.8$_{\pm0.3}$	1.7$_{\pm0.4}$	3.3$_{\pm0.6}$	**5.3**$_{\pm0.6}$
LongQA	38.1$_{\pm6.7}$	36.8$_{\pm3.7}$	**36.9**$_{\pm4.6}$	36.1$_{\pm6.7}$	27.2$_{\pm2.0}$	36.3$_{\pm3.0}$	37.0$_{\pm4.1}$	34.3$_{\pm2.2}$	34.6$_{\pm2.2}$	14.8$_{\pm2.5}$	16.9$_{\pm6.8}$	29.9$_{\pm4.9}$	**35.9**$_{\pm2.4}$
Summ.	8.2$_{\pm1.8}$	**6.7**$_{\pm2.2}$	5.6$_{\pm1.0}$	6.2$_{\pm2.5}$	3.5$_{\pm0.1}$	4.6$_{\pm1.4}$	6.3$_{\pm1.3}$	2.7$_{\pm0.6}$	3.0$_{\pm0.2}$	**11.1**$_{\pm3.0}$	10.4$_{\pm0.4}$	3.9$_{\pm1.3}$	5.1$_{\pm2.8}$
Avg. ↑	30.2$_{\pm1.9}$	25.0$_{\pm1.7}$	26.5$_{\pm1.8}$	27.6$_{\pm2.2}$	22.1$_{\pm0.9}$	**27.7**$_{\pm1.3}$	28.8$_{\pm0.8}$	22.8$_{\pm0.5}$	24.7$_{\pm0.8}$	19.8$_{\pm1.5}$	19.2$_{\pm1.8}$	20.3$_{\pm1.2}$	**27.8**$_{\pm1.3}$

Table 11. ***LongBench*** and ***Helmet*** scores (%) of NOSA compared with all baselines (Full Results). NOSA^F uses full attention during the prefill stage and sparse attention during the decode stage, while NOSA^S adopts sparse attention for both stages.

Method	***LongBench***														***Helmet***							
	GO	TQ	NQ	QS	MU	2W	MQ	RB	HQ	TR	PR	PC	SA	Avg. ↑	Recall	RAG	ICL	Cite	Rerank	LongQA	Summ.	Avg. ↑
1B Model Full Prefill																						
FullAttn	28.0	70.8	16.0	23.7	11.5	19.5	45.2	55.0	21.0	66.5	10.5	1.0	23.55	30.2	61.3$_{\pm0.4}$	39.0$_{\pm0.9}$	60.0$_{\pm0.7}$	5.5$_{\pm0.3}$	14.3$_{\pm0.0}$	17.2$_{\pm0.3}$	0.4$_{\pm0.2}$	28.3$_{\pm0.3}$
ShadKV$_{64}$	25.2	70.1	14.9	23.0	10.7	18.8	**43.4**	54.6	19.2	67.0	12.0	0.0	**22.6**	29.4	25.9$_{\pm1.0}$	37.5$_{\pm1.3}$	55.4$_{\pm0.5}$	2.7$_{\pm0.4}$	5.8$_{\pm0.0}$	16.4$_{\pm0.3}$	0.1$_{\pm0.1}$	20.6$_{\pm0.2}$
ShadKV$_{8}$	25.0	70.3	**15.4**	**23.1**	**11.1**	**19.7**	42.3	55.2	**19.6**	66.5	11.0	**1.0**	**22.6**	29.4	**48.9**$_{\pm0.8}$	38.4$_{\pm1.3}$	59.0$_{\pm0.4}$	**4.1**$_{\pm0.3}$	11.0$_{\pm0.0}$	16.7$_{\pm0.3}$	0.1$_{\pm0.1}$	**25.5**$_{\pm0.3}$
ArkVale	8.2	58.7	8.4	11.4	9.5	13.8	25.2	38.9	10.5	60.5	9.8	**1.0**	11.6	20.6	26.7$_{\pm1.6}$	34.3$_{\pm1.0}$	57.5$_{\pm0.3}$	1.1$_{\pm1.0}$	1.8$_{\pm0.4}$	16.9$_{\pm0.7}$	0.2$_{\pm0.1}$	19.8$_{\pm0.3}$
DMA^F	23.8	79.2	13.5	22.4	7.9	15.3	38.0	54.4	17.3	**68.5**	10.0	0.5	21.3	28.6	15.9$_{\pm1.0}$	39.6$_{\pm1.1}$	54.7$_{\pm0.4}$	2.7$_{\pm0.1}$	8.7$_{\pm0.0}$	15.8$_{\pm0.7}$	**0.8**$_{\pm0.3}$	19.7$_{\pm0.3}$
NOSA^F	**28.8**	**79.8**	12.9	22.9	8.3	14.5	41.8	**55.6**	18.7	68.0	**14.5**	**1.0**	20.3	**29.8**	34.3$_{\pm1.0}$	**39.8**$_{\pm1.1}$	**59.2**$_{\pm0.6}$	3.5$_{\pm0.1}$	**12.7**$_{\pm0.0}$	**17.6**$_{\pm1.0}$	**0.8**$_{\pm0.2}$	24.0$_{\pm0.3}$
1B Model Sparse Prefill																						
InfLLMv2	28.7	73.3	15.0	23.1	8.7	19.8	42.7	55.5	18.6	67.5	9.5	2.0	23.0	29.8	37.2$_{\pm1.4}$	37.2$_{\pm0.6}$	58.0$_{\pm0.1}$	4.1$_{\pm0.6}$	15.5$_{\pm0.0}$	17.7$_{\pm0.3}$	0.8$_{\pm0.3}$	24.4$_{\pm0.1}$
ShadKV$^M_{64}$	26.0	73.2	15.0	**23.8**	8.1	**18.9**	42.0	54.0	**19.3**	66.5	10.5	1.5	20.5	29.2	26.4$_{\pm0.9}$	37.9$_{\pm1.2}$	50.1$_{\pm0.3}$	3.4$_{\pm0.0}$	9.6$_{\pm0.0}$	17.3$_{\pm0.5}$	0.2$_{\pm0.1}$	20.7$_{\pm0.2}$
ShadKV$^M_{8}$	25.3	73.1	15.7	23.6	9.1	18.6	42.1	54.0	19.1	66.5	9.5	**2.0**	**21.9**	29.3	**41.9**$_{\pm0.6}$	38.5$_{\pm0.9}$	53.9$_{\pm0.4}$	**4.2**$_{\pm0.2}$	11.2$_{\pm0.0}$	**17.5**$_{\pm0.5}$	0.1$_{\pm0.1}$	**23.9**$_{\pm0.2}$
InfLLM$_{128}$	27.9	67.3	14.7	23.2	9.2	17.5	42.0	52.6	19.0	63.5	5.0	1.5	19.4	27.9	4.4$_{\pm0.3}$	34.4$_{\pm0.3}$	48.8$_{\pm0.5}$	1.2$_{\pm0.3}$	2.6$_{\pm0.0}$	13.2$_{\pm0.3}$	0.2$_{\pm0.1}$	15.0$_{\pm0.1}$
InfLLM$_{64}$	28.0	68.2	**15.3**	22.7	8.8	17.9	**43.0**	52.4	17.8	64.5	5.5	1.0	18.6	28.0	3.7$_{\pm0.3}$	36.4$_{\pm0.4}$	51.7$_{\pm0.3}$	1.1$_{\pm0.2}$	2.0$_{\pm0.0}$	13.0$_{\pm0.6}$	0.4$_{\pm0.2}$	15.5$_{\pm0.2}$
DMA^S	23.8	77.2	13.1	22.1	7.3	15.7	37.9	52.8	17.1	67.0	10.5	0.5	19.4	28.0	14.5$_{\pm0.8}$	28.2$_{\pm0.7}$	48.7$_{\pm0.3}$	2.8$_{\pm0.3}$	7.6$_{\pm0.0}$	14.6$_{\pm0.7}$	0.4$_{\pm0.1}$	16.7$_{\pm0.1}$
NOSA^S	**28.3**	**81.0**	14.2	22.7	**9.9**	15.3	42.9	**54.6**	17.9	**68.5**	**16.5**	1.5	20.2	**30.3**	29.2$_{\pm1.0}$	**39.6**$_{\pm0.3}$	**57.5**$_{\pm0.4}$	3.6$_{\pm0.4}$	**13.6**$_{\pm0.0}$	16.2$_{\pm0.7}$	**0.6**$_{\pm0.1}$	22.9$_{\pm0.2}$
3B Model Full Prefill																						
FullAttn	31.1	85.8	20.5	23.8	15.7	27.0	50.4	49.4	31.0	74.0	76.5	2.5	6.9	38.0	48.4$_{\pm1.1}$	50.6$_{\pm0.2}$	49.5$_{\pm0.5}$	9.0$_{\pm0.2}$	23.1$_{\pm0.0}$	21.5$_{\pm0.6}$	5.0$_{\pm0.2}$	29.6$_{\pm0.3}$
ShadKV$_{64}$	27.5	84.2	20.3	23.3	14.6	24.3	49.3	48.4	31.5	73.5	64.0	**2.5**	6.4	36.1	23.1$_{\pm1.2}$	48.3$_{\pm0.6}$	46.4$_{\pm0.6}$	4.4$_{\pm0.4}$	11.6$_{\pm0.0}$	20.6$_{\pm0.4}$	6.2$_{\pm0.4}$	22.9$_{\pm0.3}$
ShadKV$_{8}$	27.4	83.8	20.1	23.3	14.5	24.2	48.6	49.7	31.1	73.5	75.0	**2.5**	5.9	36.9	39.4$_{\pm0.7}$	49.0$_{\pm0.3}$	48.0$_{\pm0.9}$	5.5$_{\pm0.4}$	15.0$_{\pm0.0}$	21.2$_{\pm0.8}$	**7.3**$_{\pm0.2}$	26.5$_{\pm0.3}$
ArkVale	30.3	86.1	**20.5**	23.1	15.1	27.7	50.0	48.1	31.7	74.0	**77.0**	**2.5**	6.9	**37.9**	28.4$_{\pm0.9}$	**50.4**$_{\pm0.3}$	**49.3**$_{\pm0.6}$	5.9$_{\pm0.5}$	21.1$_{\pm0.6}$	22.1$_{\pm0.1}$	5.7$_{\pm0.6}$	26.1$_{\pm0.2}$
DMA^F	28.1	85.6	18.1	21.7	9.1	29.1	40.4	48.1	30.1	69.5	22.0	0.5	4.1	33.5	12.6$_{\pm0.1}$	47.2$_{\pm0.7}$	47.6$_{\pm1.2}$	2.8$_{\pm0.3}$	16.0$_{\pm0.0}$	**23.9**$_{\pm0.7}$	4.4$_{\pm0.1}$	22.1$_{\pm0.3}$
NOSA^F	**31.7**	**87.2**	20.2	**23.7**	**18.1**	**32.7**	**52.1**	**50.2**	**41.2**	**75.5**	45.0	1.0	**7.2**	37.4	**45.1**$_{\pm0.3}$	49.8$_{\pm0.3}$	47.7$_{\pm0.3}$	**6.3**$_{\pm0.4}$	**24.3**$_{\pm0.0}$	21.5$_{\pm0.8}$	2.7$_{\pm0.3}$	**28.2**$_{\pm0.1}$
3B Model Sparse Prefill																						
InfLLMv2	31.3	87.2	19.7	23.2	14.7	23.8	49.4	48.4	30.1	73.5	83.0	3.0	7.6	38.1	39.7$_{\pm0.6}$	49.4$_{\pm0.4}$	48.6$_{\pm0.9}$	5.7$_{\pm1.3}$	22.7$_{\pm0.0}$	23.0$_{\pm1.1}$	3.2$_{\pm0.5}$	27.5$_{\pm0.5}$
ShadKV$^M_{64}$	28.2	84.6	19.7	23.1	14.2	25.7	48.1	47.2	30.5	72.0	62.5	**3.0**	5.9	35.7	27.0$_{\pm0.6}$	48.0$_{\pm1.1}$	39.1$_{\pm0.7}$	5.0$_{\pm0.9}$	13.5$_{\pm0.2}$	21.2$_{\pm1.1}$	**5.6**$_{\pm0.4}$	22.8$_{\pm0.6}$
ShadKV$^M_{8}$	27.9	84.5	19.4	23.1	14.1	25.4	48.0	48.5	29.9	72.5	**67.0**	**3.0**	6.2	36.1	**43.7**$_{\pm0.5}$	48.9$_{\pm1.0}$	40.5$_{\pm0.6}$	7.4$_{\pm0.7}$	17.1$_{\pm0.0}$	21.1$_{\pm1.0}$	**5.6**$_{\pm0.7}$	26.3$_{\pm0.3}$
InfLLM$_{128}$	30.1	83.5	18.5	22.3	14.0	27.3	48.5	47.5	33.2	71.0	37.0	2.5	6.4	34.0	5.5$_{\pm1.4}$	44.2$_{\pm0.4}$	35.6$_{\pm0.5}$	1.7$_{\pm0.4}$	2.8$_{\pm0.0}$	12.7$_{\pm1.8}$	**5.6**$_{\pm0.4}$	15.4$_{\pm0.3}$
InfLLM$_{64}$	29.9	82.8	17.9	22.3	15.6	26.6	47.9	46.9	31.9	73.0	41.5	1.5	5.8	34.1	4.5$_{\pm0.4}$	44.1$_{\pm0.5}$	37.5$_{\pm0.2}$	1.5$_{\pm0.5}$	1.9$_{\pm0.0}$	12.0$_{\pm1.9}$	4.9$_{\pm0.6}$	15.2$_{\pm0.2}$
DMA^S	28.1	85.6	18.1	21.7	9.1	29.1	40.4	48.1	30.1	69.5	22.0	0.5	4.1	31.3	12.3$_{\pm0.6}$	41.2$_{\pm0.9}$	41.1$_{\pm0.0}$	3.6$_{\pm0.4}$	10.1$_{\pm0.0}$	21.5$_{\pm0.8}$	5.3$_{\pm0.6}$	19.3$_{\pm0.2}$
NOSA^S	**31.7**	**88.2**	**21.2**	**23.5**	**19.7**	**32.9**	**52.5**	**49.4**	**40.0**	**74.5**	52.5	0.6	**7.2**	**38.0**	36.1$_{\pm1.0}$	**49.2**$_{\pm0.6}$	**52.2**$_{\pm0.4}$	**6.4**$_{\pm0.2}$	**26.4**$_{\pm0.0}$	**21.6**$_{\pm0.7}$	2.6$_{\pm1.1}$	**27.8**$_{\pm0.3}$
8B Model Full Prefill																						
FullAttn	31.0	88.1	19.4	25.0	27.4	40.6	52.3	58.3	50.0	74.5	94.0	3.0	6.6	43.9	89.5$_{\pm0.3}$	56.1$_{\pm1.0}$	61.3$_{\pm0.4}$	14.5$_{\pm0.4}$	39.8$_{\pm0.0}$	26.6$_{\pm0.6}$	10.7$_{\pm1.0}$	42.6$_{\pm0.3}$
ShadKV$_{64}$	28.5	87.8	**19.4**	24.2	25.0	41.1	52.0	57.4	48.7	**74.5**	89.5	3.0	**6.2**	42.9	55.3$_{\pm0.2}$	55.6$_{\pm0.9}$	58.9$_{\pm0.6}$	10.7$_{\pm0.6}$	21.3$_{\pm0.1}$	25.2$_{\pm1.2}$	7.8$_{\pm0.5}$	33.5$_{\pm0.3}$
ShadKV$_{8}$	28.8	87.7	**19.4**	24.4	26.2	**41.4**	51.4	57.5	48.8	**74.5**	**93.0**	3.0	6.1	43.3	78.3$_{\pm0.4}$	55.7$_{\pm1.1}$	60.6$_{\pm0.5}$	12.4$_{\pm0.9}$	29.3$_{\pm0.0}$	26.6$_{\pm0.8}$	**8.8**$_{\pm1.3}$	38.8$_{\pm0.6}$
ArkVale	29.7	**88.4**	17.6	24.6	25.0	33.7	36.2	45.9	42.7	71.0	92.5	3.0	3.4	39.5	56.2$_{\pm1.5}$	55.6$_{\pm0.8}$	61.5$_{\pm0.6}$	10.7$_{\pm2.4}$	**35.1**$_{\pm0.1}$	26.5$_{\pm1.5}$	8.2$_{\pm0.8}$	36.2$_{\pm0.4}$
DMA^F	27.6	86.7	16.7	23.5	24.4	39.0	47.1	55.1	47.8	73.0	69.0	**6.0**	5.7	40.1	33.1$_{\pm0.2}$	52.4$_{\pm0.6}$	**63.8**$_{\pm0.1}$	10.4$_{\pm0.7}$	26.5$_{\pm0.0}$	25.7$_{\pm1.9}$	6.5$_{\pm0.7}$	31.2$_{\pm0.5}$
NOSA^F	**31.8**	86.8	15.2	**25.1**	**29.2**	41.1	**53.0**	**59.7**	**50.7**	74.0	92.0	4.0	5.4	**43.7**	**86.3**$_{\pm0.0}$	**56.3**$_{\pm0.4}$	60.6$_{\pm0.5}$	**13.6**$_{\pm1.1}$	30.8$_{\pm0.0}$	**26.7**$_{\pm1.5}$	8.5$_{\pm0.7}$	**40.4**$_{\pm0.5}$
8B Model Sparse Prefill																						
InfLLMv2	32.4	86.6	17.7	25.0	29.3	42.2	52.9	59.5	48.7	75.5	95.5	4.5	5.8	44.3	72.7$_{\pm0.2}$	54.1$_{\pm1.1}$	68.9$_{\pm0.3}$	13.8$_{\pm0.7}$	37.5$_{\pm0.0}$	26.6$_{\pm1.5}$	8.6$_{\pm0.6}$	40.3$_{\pm0.3}$
ShadKV$^M_{64}$	28.3	89.5	18.1	24.0	23.6	37.8	**52.9**	56.6	**49.0**	72.5	74.5	3.0	4.9	41.1	54.5$_{\pm0.9}$	53.1$_{\pm0.7}$	52.4$_{\pm0.5}$	10.7$_{\pm1.1}$	23.4$_{\pm0.0}$	23.9$_{\pm1.1}$	8.1$_{\pm0.6}$	32.3$_{\pm0.4}$
ShadKV$^M_{8}$	28.3	**89.6**	17.4	23.8	24.3	37.6	52.4	56.7	48.6	72.5	75.5	2.5	4.8	41.1	**74.6**$_{\pm0.2}$	53.1$_{\pm0.7}$	53.5$_{\pm0.2}$	11.6$_{\pm0.5}$	28.9$_{\pm0.0}$	25.9$_{\pm2.1}$	7.7$_{\pm1.0}$	36.5$_{\pm0.4}$
InfLLM$_{128}$	30.4	87.0	18.7	22.8	31.8	40.7	50.1	**60.6**	48.2	71.5	49.0	**7.5**	3.4	40.1	5.8$_{\pm0.1}$	47.2$_{\pm0.5}$	36.7$_{\pm0.9}$	1.7$_{\pm0.1}$	2.2$_{\pm0.0}$	16.6$_{\pm2.5}$	8.1$_{\pm0.4}$	16.9$_{\pm0.3}$
InfLLM$_{64}$	30.2	86.2	**19.0**	23.1	**33.8**	38.8	50.2	60.4	**49.0**	**75.0**	57.5	5.5	3.9	41.0	5.5$_{\pm0.5}$	47.8$_{\pm1.2}$	37.4$_{\pm0.8}$	1.4$_{\pm0.5}$	2.4$_{\pm0.0}$	13.5$_{\pm3.9}$	7.0$_{\pm1.2}$	16.4$_{\pm0.7}$
DMA^S	27.9	87.2	15.0	23.1	17.9	36.6	45.1	53.9	44.4	73.0	35.5	2.5	4.4	35.9	22.7$_{\pm0.7}$	39.9$_{\pm0.7}$	56.1$_{\pm0.4}$	9.3$_{\pm0.9}$	21.6$_{\pm0.0}$	**28.2**$_{\pm1.8}$	6.5$_{\pm0.8}$	26.3$_{\pm0.3}$
NOSA^S	**32.6**	86.8	16.5	**24.4**	29.1	40.8	**52.9**	59.0	48.3	74.0	**89.0**	4.0	**5.2**	**43.3**	67.2$_{\pm0.8}$	**54.3**$_{\pm1.2}$	**65.1**$_{\pm0.5}$	**12.5**$_{\pm1.3}$	**31.0**$_{\pm0.0}$	25.1$_{\pm1.8}$	**8.2**$_{\pm0.7}$	**37.6**$_{\pm0.7}$