

NOSA: Native and Offloadable Sparse Attention

Yuxiang Huang^{1*}, Pengjie Wang^{1*}, Jicheng Han¹, Weilin Zhao¹,
Zhou Su², Ao Sun¹, Hongya Lyu², Hengyu Zhao²,
Yudong Wang^{1,2}, Chaojun Xiao^{1†}, Xu Han^{1,3†}, Zhiyuan Liu^{1,3†},

¹NLP Group, DCST, IAI, BNRIST, Tsinghua University, China. ²OpenBMB, China.

³Jiangsu Collaborative Innovation Center for Language Ability, Jiangsu Normal Univ., China.
{huang-yx21, wpj24}@mails.tsinghua.edu.cn, {xcj, han-xu, liuzy}@tsinghua.edu.cn

Abstract

Decoding throughput gains from larger inference batches are often limited by GPU memory, mainly due to the large size of KV cache. Training-free KV cache offloading mitigates this issue by storing redundant context on the CPU and fetching sparse KV subsets, but often harms long-generation quality due to training-inference mismatch of sparse patterns. Trainable sparse attention can improve quality, yet its unconstrained KV accesses are inefficient for offloading and may incur excessive CPU–GPU transfers. To this end, we propose NOSA, a trainable sparse attention mechanism designed for KV cache offloading. NOSA explicitly bounds CPU–GPU KV transfers, enabling low communication overhead and high decoding throughput. We further develop NOSI, an offloading inference system that realizes these efficiency gains. Experiments on {1, 3, 8}B LLMs show that NOSA outperforms KV cache offloading baselines on general, long-input, and long-generation tasks, while improving decoding throughput by up to 5.04×, 1.92×, and 1.83× over FULLATTN, INFLLMV2, and SHADOWKV, respectively. Code available at <https://github.com/thunlp/NOSA>.

1 Introduction

Decoding efficiency of large language models (LLMs) generation (OpenAI, 2025; Anthropic, 2025; DeepSeek-AI, 2025b) 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, 2025a; 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 key-value (KV) cache offloading is

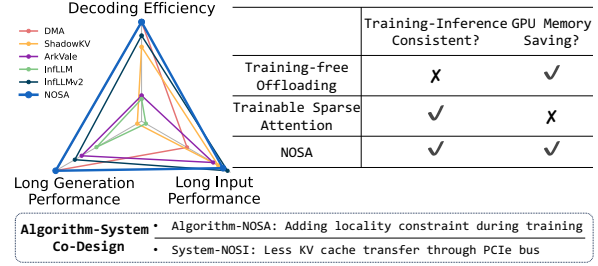


Figure 1: Overview of NOSA and NOSI.

widely adopted (Xiao et al., 2024a; Sun et al., 2025; Yang et al., 2025). These methods keep most 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 where selection errors accumulate.

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 to advance decoding acceleration with minimal performance degradation. We summarize our contributions as follows.

* indicates equal contribution.

† indicates corresponding authors.

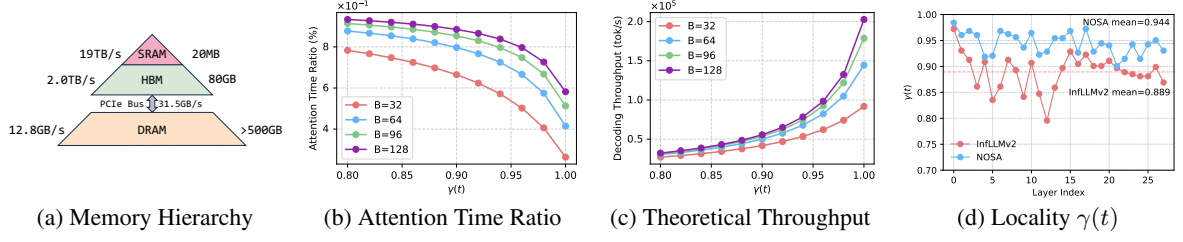


Figure 2: The memory hierarchy of NVIDIA A800-80GB, the attention time ratio, the theoretical decoding throughput with respect to various cache hit rates, and INFLLMv2’s locality. “B” represents the batch size.

Contribution 1: We first analyze the feasibility of building KV cache offloading systems based on trainable sparse attention methods. Using INFLLMv2 (Zhao et al., 2025) as the representative of trainable sparse attention, we identify strong block-selection locality and theoretically analyze the resulting PCIe bottleneck. We show that trainable sparse attention naturally supports KV cache offloading, yet PCIe bandwidth limitations can make decoding communication-bound.

Contribution 2: Building on these, we introduce NOSA, a trainable sparse attention mechanism natively designed for offloading that explicitly limits KV cache transmission volume. NOSA decomposes KV selection into query-aware and query-agnostic components, applying eviction to the latter to bound CPU-fetched KV blocks. We train NOSA-integrated models and evaluate them on short and long context tasks, showing strong performance and consistent gains over KV offloading baselines, especially for long generations.

Contribution 3: Vanilla offloading implementations are insufficient to fully reveal PCIe bottlenecks. We therefore develop NOSI, an offloading system tailored for NOSA. With system-level optimizations, NOSI achieves substantially higher decoding efficiency across settings, improving throughput by up to $5.04\times$, $1.92\times$, and $1.83\times$ 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 $\mathbf{Q}$, key $\mathbf{K}$, and value $\mathbf{V}$ matrices are then obtained through linear projections. The attention computation is then formulated as below, where

$\mathbf{M} \in \{0, -\infty\}^{n \times n}$ is the attention mask.

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

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 $f_c: \mathbb{R}^{n_b \times d} \rightarrow \mathbb{R}^{1 \times d}$. 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 in Eq. (1) is computed with a mask $\mathbf{M} = (m_{ij})_{n \times n}$ that

$$m_{ij} = \begin{cases} 0, & \mathbb{I}_{\text{causal}} \wedge (\mathbb{I}_{\text{sink}} \vee \mathbb{I}_{\text{window}} \vee \mathbb{I}_{\text{topk}}); \\ -\infty, & \text{otherwise}; \end{cases} \quad (2)$$

$$\mathbb{I}_{\text{causal}} = i \geq j; \quad \mathbb{I}_{\text{sink}} = j < n_s; \quad \mathbb{I}_{\text{window}} = i - j < n_w;$$

$$\mathbb{I}_{\text{topk}} = s_{ij}^q \in \text{Top}_k(s_{i, \leq \lfloor i/n_b \rfloor}^q).$$

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; 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 according attention scores. 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, an importance score from a scoring function $s_i^e = f_e(h_i) \in \mathbb{R}$ is first calculated, and then select the top- k positions. Query-agnostic selection cannot recall evicted entries and therefore yields an eviction pattern.

2.2 Observations

We conduct an analytical study on the potential of applying trainable sparse attention mechanisms to offloading systems on typical compute architectures (as illustrated in Figure 2a). Based on this, we derive the following key observations.

Observation 1: Locality in Query-aware Selection

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

We first analyze the locality of KV selection in a representative trainable sparse attention mechanism, INFLLMV2 (Zhao et al., 2025). Since KV cache offloading is primarily constrained by PCIe transmission overhead, higher locality allows caching and reusing KV caches selected in the previous decoding step for the current step, thereby reducing communication cost.

Definition 2.1 (Locality). Given an attention mask $M = (m_{ij})_{n \times n}$, we define the set of selected tokens at the t -th step as $\Gamma(t) = \{i : m_{it} = 0\}$. The locality at decoding step t is defined as the fraction of tokens selected at step t and also selected at step $t - 1$: $\gamma(t) = \frac{|\Gamma(t) \cap \Gamma(t-1)|}{|\Gamma(t)|}$.

We measure the locality $\gamma(t)$ for each transformer layer using an input sequence of 16K tokens, among which 4K tokens are selected. Figure 2d shows that, we have $\gamma(t) \geq 0.8$ for most layers. This strong locality implies that fewer than 20% of the KV entries change per step and thus require communication, making KV cache offloading possible. Notably, this inherent locality emerges naturally from the trained sparse attention without any explicit constraints.

Observation 2: PCIe Communication Bound

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

Although Observation 1 indicates that high locality enables the potential for offloading by reusing previously selected tokens as cache, we find that a $\sim 80\%$ locality is still insufficient to effectively mitigate PCIe communication overhead. Figure 2b shows that more than 80% of the decoding time is spent within the attention mechanism, suggesting that decoding throughput remains constrained by

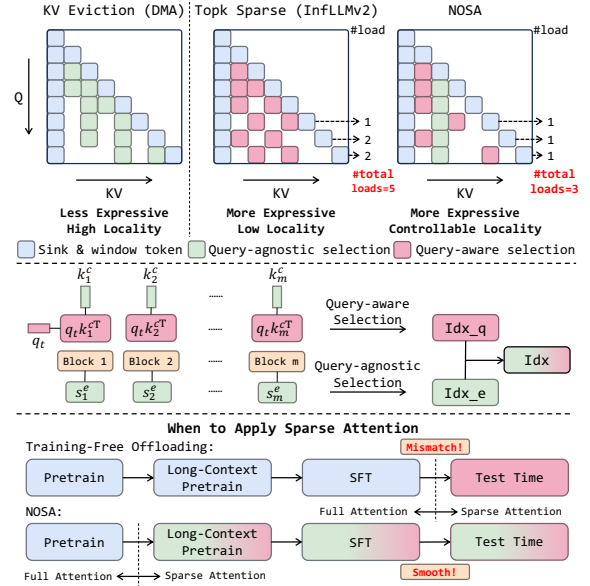


Figure 3: The framework of NOSA. See Section 3 for more details on NOSA’s design.

communication.

To further analyze this effect, we simulate the theoretical inference latency and decoding throughput under varying cache hit rates (i.e., locality $\gamma(t)$), following the efficiency estimation protocol proposed in Yuan et al. (2024). Figure 2c shows that increasing the cache hit rate significantly reduces the fraction of time spent on attention, thereby improving the theoretical decoding throughput.

Takeaway: Building on Observation 1 and 2, achieving high decoding throughput critically requires enforcing a training-time constraint that promotes higher cache hit rates through natively offloadable sparse attention.

3 NOSA

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 Offloadable Trainable Sparse Attention

Adding Locality Constraint. As discussed in Sec. 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

$$\forall t \in \{2, \dots, n\}, \gamma(t) \geq \gamma_0. \quad (3)$$

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 Sec. 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 $\Gamma_e(t') \subseteq \Gamma_e(t) \cup \{t+1, \dots, t'\}$, 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 $\Gamma_e(t) = \text{ArgTop}_k(\{s_i^e\}_{i=0}^t)$. 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

$$s_j^e = \tau(\mathbf{v}_j \mathbf{W}_1) \mathbf{W}_2, \quad (4)$$

where τ is a non-linear function, $\mathbf{W}_1 \in \mathbb{R}^{d_{kv} \times n_{kv}}$, $\mathbf{W}_2 \in \mathbb{R}^{n_{kv} \times 1}$ are trainable parameters, and d_{kv}, n_{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 Sec. 3.2. 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, 2025b) 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

(Sec. 3.2). 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, followed by a max pooling to each block. We denote the j -th sub-block as $\langle j \rangle$, and proceed as follows:

$$s_{i\langle j \rangle}^{\{q,e\}} = \text{mean}_{t \in \langle j \rangle} s_{it}^{\{q,e\}}, s_{i[j]}^{\{q,e\}} = \max_{\langle t \rangle \subset [j]} s_{i\langle t \rangle}^{\{q,e\}}. \quad (5)$$

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 $s_{i[j]}^q$. We then set these places to infinity and select the query-agnostic blocks by $s_{i[j]}^e$. Such process follows taking a Top- k selection ($\Gamma(i) = \text{ArgTop}_{k/n_b}(\{s_{i[j]}^e\}_{j=1}^{\lfloor t/n_b \rfloor})$) on the following score:

$$s_{i[j]} = \begin{cases} +\infty, & s_{i[j]}^q \in \text{Top}_{k_q/n_b}(\{s_{i[j]}^q\}_{j=1}^{\lfloor t/n_b \rfloor}); \\ s_{i[j]}^e, & \text{otherwise.} \end{cases} \quad (6)$$

This design inherently guarantees a lower bound on locality, as formalized below (Proved in App. E).

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

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 $\mathbf{S}^e = (s_j^e)_t$ according to Eq. (4). Next, we utilize the selection process defined in Eq. (6) 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 $b_j = s_j^e = \tau(\mathbf{v}_j \mathbf{W}_1) \mathbf{W}_2$.

Finally, the attention output $\mathbf{o}_t$ is computed as

$$\mathbf{o}_t = \sum_{j=1}^t \frac{\exp(\mathbf{q}_t \mathbf{k}_j^\top + m_j + b_j)}{\sum_{i=1}^t \exp(\mathbf{q}_t \mathbf{k}_i^\top + m_i + b_i)} \mathbf{v}_j. \quad (7)$$

We observe that the query-agnostic selection is highly sensitive to numerical precision. Therefore,

Impl.	INFLLMv2	LOCRET	DMA	S-DMA	ED-DMA
RULER (%) $\uparrow$	60.3	55.9	56.2	60.9	61.9

Table 1: Ablating of eviction head implementations.

we omit the exponential (exp) operation before token selection; that is, we select the top- k indices based on pre-exponential importance score $s_{[j]}^e$ 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 Sec. 3.2, ED-DMA yields the most stable and effective results, whereas alternative approaches lead to noticeably greater performance degradation.

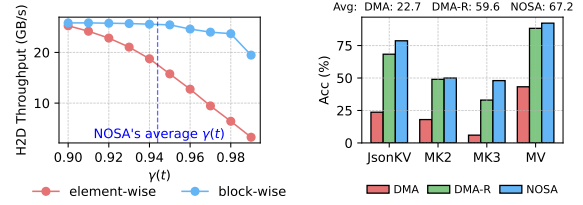
Training with NOSA. The design of NOSA is fully trainable, as the eviction head receives gradients through the attention bias b . To equip LLMs with native offloading capability, *we incorporate NOSA throughout long-context adaptation*. We pre-train with vanilla attention on short-context data, then switch to NOSA for long-context continual pretraining and the following SFT stage, therefore delivering the model with built-in NOSA support.

3.2 Ablations on Algorithm Design

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., $s_j^e = \text{MLP}(h_j)$. 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 computation is identical to vanilla attention. More details on each implementation and the full results are included in App. B.3.1.

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 1, ED-DMA outperforms all other implementations. Notably, the larger 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



(a) Block-wise vs. element-wise H2D throughput (b) Recall accuracies of DMA, DMA-R, and NOSA.

Figure 4: Comparison between DMA and NOSA.

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 the locality constraint.

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 KVs in a block-wise manner; and (2) NOSA uses both query-aware and 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 two patterns while only fetching 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. Figure 4a shows that element-wise communication degrades sharply under high locality, whereas block-wise communication sustains high throughput.

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), and also include DMA-R (trained with DMA, inference with NOSA). Figure 4b shows DMA and DMA-R have lower recall scores, suggesting query-aware selection is necessary in both training and inference.

4 NOSI: Inference System for NOSA

The vanilla implementation of NOSA based on Hugging Face Transformers (Wolf et al., 2020) is highly inefficient due to inefficient kernels and excessive small kernel launch overheads. To mitigate

Algorithm 1 NOSA’s attention computation in NOSI at the t -th decoding step.

Input: Hidden state H ; weights W_{qkv}, W_1, W_2 ; cache engine; hyperparameters params, including k, k_e, k_q , block size, and pooling strides.

Output: Attention output $H' \in \mathbb{R}^{B \times 1 \times d}$.

- 1: $Q, K, V, s_t^e \leftarrow$
 fused_linear_projection($H, W_{\text{qkv}}, W_1, W_2$)
- 2: apply_rope_inplace(Q, K)
- 3: $K_c \leftarrow$ engine.update_compressed_k(K)
- 4: $S_c^q, S_c^e \leftarrow$
 engine.update_importance_score(s_t^e)
- 5: // Stage 1: select top- k KV blocks
- 6: $S_c^q \leftarrow QK_c^\top$
- 7: $q_aware_score, q_agnostic_score \leftarrow$
 fused_pooling($S_c^q, S_c^e, \text{params}$)
- 8: $\text{topk_idx} \leftarrow$ select_topk_indices(
 $q_aware_score, q_agnostic_score,$
 $\text{params}.k_e, \text{params}.k$)
- 9: // Move cache-missed KV blocks from CPU to GPU
- 10: $\text{load_pos} \leftarrow$ generate_load_positions(
 $\text{topk_idx}, \text{engine.current_mapping}$)
- 11: engine.host_to_device_communication(load_pos)
- 12: // Stage 2: compute sparse attention
- 13: $H' \leftarrow$ flash_attn_with_kvcache(
 $Q, \text{cache.key}, \text{cache.value}$)
- 14: return H'

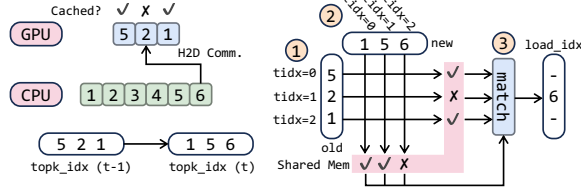


Figure 5: Memory layout and block selection.

these issues, we implement NOSI, a **NOSA**-based **S**ystem for **I**nterface, which exposes true communication overhead from other performance bottlenecks. A pseudocode is provided in Algorithm 1.

Kernel Fusion. We utilize FLASHINFER’s (Ye et al., 2025) optimized implementations for LayerNorm, RoPE, and FFN. Because the eviction heads are lightweight, executing them separately introduces non-negligible overhead; we therefore fuse them with QKV splitting into a single Triton kernel. We also fuse the max-pooling operations for query-aware and query-agnostic scores, as they share the same computation pattern. Finally, we use CUDA Graphs to reduce kernel-launch overhead from max-pooling to index selection.

Memory Layout and Block Selection. We organize the KV cache into minimal block-sized units on both GPU and CPU (Figure 5) to reduce communication overhead. Blocks not used in the current step are discarded and replace them with the

Batch Size	16	32	64
Throughput (GB/s)	17.04	20.36	22.35

Table 2: Real-world communication throughput.

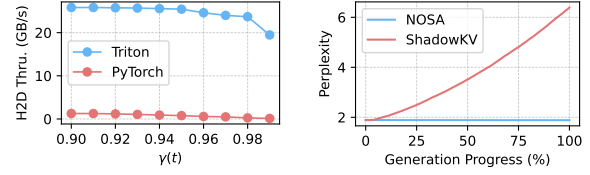


Figure 6: Throughput (↑) in H2D communication. Figure 7: Perplexity (↓) in long-generation.

newly selected ones. This requires solving a set-difference problem and producing a replacement mapping, where we use a custom CUDA kernel (Figure 5). 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 existence of new indices, and finally performs a per-thread scan to generate the final mapping.

Offloading Communication. Since NOSA selects blocks independently for each attention head, each block is relatively small. This fine-grained partitioning results in fragmented host-to-device communication and poor PCIe bandwidth utilization. To mitigate this, 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. We compare our kernel implementation with the vanilla PyTorch implementation. As shown in Figure 6, our method consistently achieves substantially higher throughput than the PyTorch baseline. Furthermore, we evaluate the communication throughput in real-world inference scenarios using an 8B model with batch sizes $B \in \{16, 32, 64\}$ on a PCIe 4.0 platform. The results, summarized in Table 2, demonstrate the efficiency and scalability of our implementation.

5 Experiments

We conduct experiments to address these questions:

- (Q1) Does NOSA achieve better performance on long-context inputs compared with other baselines?
- (Q2) Can NOSA perform competitively on general tasks and long-generation reasoning tasks?
- (Q3) Does NOSA improve decoding efficiency?

We also include experiments on length generalization (App. B.2), hyperparameter analysis (App. B.3.2), efficiency across different semantics (App. B.3.3), and different hardware (App. B.3.5).

Method	LongBench													Helmet								
	GO	TQ	NQ	QS	MU	2W	MQ	RB	HQ	TR	PR	PC	SA	Avg. $\uparrow$	Recall	RAG	ICL	Cite	Rerank	LongQA	Summ.	Avg. $\uparrow$
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	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	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	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	86.3 $\pm$ 0.0	56.3 $\pm$ 0.4	60.6 $\pm$ 0.5	13.6 $\pm$ 1.1	30.8 $\pm$ 0.0	26.7 $\pm$ 1.5	8.5 $\pm$ 0.7	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 ₆₄ ^M	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 ₈ ^M	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 $\pm$ 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	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	54.3 $\pm$ 1.2	65.1 $\pm$ 0.5	12.5 $\pm$ 1.3	31.0 $\pm$ 0.0	25.1 $\pm$ 1.8	8.2 $\pm$ 0.7	37.6 $\pm$ 0.7

Table 3: *LongBench*¹ and *HELMET* scores (%) of NOSA compared with all baselines. ^F represents full prefill, sparse decode; ^S represents sparse prefill and decode. Results for 1B and 3B models are reported in Table 8.

Method	Recall	RAG	ICL	Cite	Rerank	LongQA	Summ.	Avg. $\uparrow$
FULLATTN	21.7 $\pm$ 0.8	38.1 $\pm$ 3.9	58.0 $\pm$ 0.8	2.9 $\pm$ 0.3	3.3 $\pm$ 1.0	13.9 $\pm$ 2.1	4.9 $\pm$ 1.7	20.4 $\pm$ 0.6
INFLLMv2	12.0 $\pm$ 0.6	30.9 $\pm$ 3.8	49.0 $\pm$ 4.6	5.4 $\pm$ 0.9	0.5 $\pm$ 0.3	5.0 $\pm$ 1.4	1.5 $\pm$ 0.2	14.9 $\pm$ 0.9
INFLLM	11.2 $\pm$ 0.8	36.5 $\pm$ 2.9	35.3 $\pm$ 3.4	6.1 $\pm$ 1.5	0.6 $\pm$ 0.4	6.1 $\pm$ 0.8	12.3 $\pm$ 4.0	15.4 $\pm$ 0.7
SHADKV ₆₄	10.0 $\pm$ 1.4	41.1 $\pm$ 2.5	52.7 $\pm$ 1.9	1.6 $\pm$ 0.4	0.0 $\pm$ 0.0	3.9 $\pm$ 5.5	2.1 $\pm$ 0.7	15.9 $\pm$ 0.8
SHADKV ₈	13.4 $\pm$ 1.2	38.7 $\pm$ 3.7	56.7 $\pm$ 2.1	3.6 $\pm$ 0.2	0.8 $\pm$ 0.2	4.4 $\pm$ 2.1	0.5 $\pm$ 0.7	16.9 $\pm$ 0.6
SHADKV ₆₄ ^M	10.4 $\pm$ 1.1	32.9 $\pm$ 5.8	58.3 $\pm$ 1.3	1.9 $\pm$ 0.1	0.0 $\pm$ 0.0	17.8 $\pm$ 11.0	3.9 $\pm$ 1.4	17.9 $\pm$ 1.5
SHADKV ₈ ^M	14.7 $\pm$ 0.7	33.6 $\pm$ 4.9	63.7 $\pm$ 4.2	3.0 $\pm$ 1.1	0.1 $\pm$ 0.1	11.1 $\pm$ 6.3	0.0 $\pm$ 0.0	18.0 $\pm$ 0.6
DMA	3.1 $\pm$ 1.9	34.2 $\pm$ 4.1	54.3 $\pm$ 3.8	3.0 $\pm$ 2.3	1.3 $\pm$ 0.6	1.7 $\pm$ 2.4	1.5 $\pm$ 1.2	14.2 $\pm$ 1.4
DMA ^S	3.4 $\pm$ 2.1	29.5 $\pm$ 4.2	52.3 $\pm$ 6.7	5.1 $\pm$ 1.1	0.1 $\pm$ 0.2	6.1 $\pm$ 4.2	4.2 $\pm$ 2.6	14.4 $\pm$ 1.3
NOSA	13.2 $\pm$ 1.2	34.1 $\pm$ 5.4	68.3 $\pm$ 1.3	3.8 $\pm$ 0.4	0.7 $\pm$ 0.2	12.2 $\pm$ 7.5	10.6 $\pm$ 6.0	20.4 $\pm$ 1.1

Table 4: *HELMET* benchmark results (%) at 128K.

5.1 Experimental Setup

We briefly describe the major experimental settings here. Additional details can be found in App. C.

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

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 App. C.2.1.

Baselines. INFLLMv2 and FULLATTN are treated as the upper bound of our method. We include SHADKV (Sun et al., 2025) (abbreviated as SHADKV), INFLLM (Xiao et al., 2024a), and ARKVALE (Chen et al., 2024a) as our primary training-free offloading baselines. We also include VLLM (Kwon et al., 2023) and SGLANG (Zheng et al., 2024) in the efficiency benchmark. Please refer to App. C.2.2 and D for detailed settings.

¹See App. C.3 for *LongBench* task abbreviations.

Model	Method	MM	MMP	BB	GS	MA	DR	MB	HE	Avg. $\uparrow$
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	37.8	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 *general tasks*². All models are evaluated with full attention inference.

5.2 Long Context Tasks

To answer **Q1**, we evaluate NOSA against the baselines on LongBench and HELMET (Table 3).

NOSA achieves the highest performance on long-context inputs compared with all baselines. Table 3 shows that NOSA achieves stronger task performance compared to baselines under both full prefill and sparse prefill settings, with no significant degradation relative to INFLLMv2 and FULLATTN.

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.

NOSA achieves consistently good performance at longer input lengths. To evaluate whether NOSA can achieve better performance under long-

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)

Table 7: Decoding throughput (tok/s) $\uparrow$. “EB” = equivalent batch size; “Impl.”/“Off.” = implementation/offloading. For each EB, we fix GPU KV cache size and report the real batch size in “()”. “–” indicates no feasible batch (≥ 1).

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, 2025b; Shi et al., 2025; Yan et al., 2026; 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.

Note that we omit some previous related method, e.g., MOBA, NSA, and SEERATTN in our comparison, since they are proven to be on-par or weaker than our selected baselines. INFLLMv2 is a direct successor to NSA and has been shown to outperform NSA. SSA (Shen et al., 2026) provides experimental evidence that MoBA is substantially weaker than NSA. SEERATTN is weaker

than QUEST (Tang et al., 2024) in, which is in turn weaker than SHADOWKV (Sun 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) focusing on supporting LLM serving systems with sparse attentions to enhance decoding efficiency.

7 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.

Limitations

Due to time and resource constraints, we evaluate NOSA only on models ranging from 1B to 8B parameters, and do not integrate NOSA into larger LLMs, alternative architectures, or multi-modal LLMs at this stage. Moreover, all efficiency results are obtained with NOSI rather than existing serving frameworks such as vLLM and SGLang, as integrating NOSA into these systems requires substantial engineering effort. Future work includes integrating NOSA and NOSI into mainstream LLM serving engines, accelerating rollout in LLM reinforcement learning, and evaluating NOSA on a broader range of model architectures, such as hybrid models.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (2024YFB4505603), National Natural Science Foundation of China (No. 62576186), supported by Tsinghua University Initiative Scientific Research Program. Chaojun Xiao is supported by the China National Postdoctoral Program for Innovative Talents (grant no. BX20250388).

References

- Anthropic. 2025. Claude-sonnet-4.5.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.
- Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, and 1 others. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of ACL*, pages 3119–3137.
- Adithya Bhaskar, Alexander Wettig, Tianyu Gao, Yihe Dong, and Danqi Chen. 2025. Cache me if you can: How many kvs do you need for effective long-context lms? *arXiv preprint arXiv:2506.17121*.
- Ziyi Cao, Qingyi Si, Jingbin Zhang, and Bingquan Liu. 2025. Sparse attention across multiple-context kv cache. *arXiv preprint arXiv:2508.11661*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.
- Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. 2024a. Arkvale: Efficient generative llm inference with recallable key-value eviction. *Proceedings of NeurIPS*.
- Xinhang Chen, Chao Zhang, Jiahuan He, Wei Liu, Jianming Zhang, Wenlong Zhou, Xiao Li, Pai Zeng, Shiyong Li, Yuanpan Qian, and 1 others. 2025. Ess: An offload-centric latent-cache management architecture for deepseek-v3. 2-exp. *arXiv preprint arXiv:2512.10576*.
- Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and 1 others. 2024b. Magicpig: Lsh sampling for efficient llm generation. *Proceedings of ICLR*.
- Ziyang Chen, Xing Wu, Junlong Jia, Chaochen Gao, Qi Fu, Debing Zhang, and Songlin Hu. 2026. Longbench pro: A more realistic and comprehensive bilingual long-context evaluation benchmark. *arXiv preprint arXiv:2601.02872*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- DeepSeek-AI. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Nature*.
- DeepSeek-AI. 2025b. Deepseek-v3.2-exp: Boosting long-context efficiency with deepseek sparse attention.
- Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. 2024. Longrope: Extending llm context window beyond 2 million tokens. *Proceedings of ICML*.
- Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *Proceedings of NAACL*.
- Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac’h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. [The language model evaluation harness](#).

- Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and 1 others. 2025. Seerattention: Learning intrinsic sparse attention in your llms. *Proceedings of NeurIPS*.
- Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive kv cache compression for llms. *Proceedings of ICLR*.
- Nuno Gonçalves, Marcos Treviso, and André FT Martins. 2025. Adasplash: Adaptive sparse flash attention. *Proceedings of ICML*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Meng-Hao Guo, Jiajun Xu, Yi Zhang, Jiayi Song, Haoyang Peng, Yi-Xuan Deng, Xinzhi Dong, Kiyohiro Nakayama, Zhengyang Geng, Chen Wang, and 1 others. 2025. R-bench: Graduate-level multi-disciplinary benchmarks for llm & mllm complex reasoning evaluation. *Proceedings of ICML*.
- Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024a. Lm-infinite: Zero-shot extreme length generalization for large language models. *Proceedings of ACL*.
- Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024b. Lm-infinite: Simple on-the-fly length generalization for large language models. *Proceedings of NAACL*.
- Xingyang He, Jie Liu, and Shaowei Chen. 2025. Taskkv: Task-aware kv cache optimization via semantic differentiation of attention heads. *arXiv preprint arXiv:2501.15113*.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Proceedings of NeurIPS*.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Proceedings of NeurIPS*.
- Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2025. Squeezed attention: Accelerating long context length llm inference. *Proceedings of ACL*.
- Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekish, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What’s the real context size of your long-context language models? *Proceedings of COLM*.
- Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, and 1 others. 2024. Minicpm: Unveiling the potential of small language models with scalable training strategies. *Proceedings of COLM*.
- Yuxiang Huang, Binhang Yuan, Xu Han, Chaojun Xiao, and Zhiyuan Liu. 2025. Locret: Enhancing eviction in long-context llm inference with trained retaining heads on consumer-grade devices. *Proceedings of TMLR*.
- Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.
- Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. 2025. Towards economical inference: Enabling deepseek’s multi-head latent attention in any transformer-based llms. *Proceedings of ACL*.
- Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, and 1 others. 2024. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Proceedings of NeurIPS*.
- Zhonghua Jiang, Kunxi Li, Yiyun Zhou, Sihao Liu, Zhaode Wang, Shengyu Zhang, and 1 others. 2025. Purekv: Plug-and-play kv cache optimization with spatial-temporal sparse attention for vision-language large models. *arXiv preprint arXiv:2510.25600*.
- Minsoo Kim, Kyuhong Shim, Jungwook Choi, and Simyung Chang. 2024. Infinipot: Infinite context processing on memory-constrained llms. *Proceedings of EMNLP*.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of SOSP*.
- Yucheng Li, Huiqiang Jiang, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Amir H Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and 1 others. 2025. Mminference: Accelerating pre-filling for long-context vlms via modality-aware permutation sparse attention. *Proceedings of ICML*.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Proceedings of NeurIPS*.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let’s verify step by step. *Proceedings of ICLR*.

- Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, and 1 others. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.
- Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. 2024b. Minicache: Kv cache compression in depth dimension for large language models. *Proceedings of NeurIPS*.
- Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. 2025. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. *Proceedings of DAC*.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Proceedings of NeurIPS*.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024c. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *Proceedings of ICML*.
- Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Proceedings of ICLR*.
- Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, and 1 others. 2025. Moba: Mixture of block attention for long-context llms. *Proceedings of NeurIPS*.
- Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, and 1 others. 2025. Large language model agent: A survey on methodology, applications and challenges. *arXiv:2503.21460*.
- MiniCPM-Team. 2025. Minicpm4: Ultra-efficient llms on end devices. *arXiv preprint arXiv:2506.07900*.
- Junlin Mu, Hantao Huang, Jihang Zhang, Minghui Yu, Tao Wang, and Yidong Li. 2025. Sals: Sparse attention in latent space for kv cache compression. *Proceedings of NeurIPS*.
- OpenAI. 2025. Gpt-5.
- OpenBMB. 2025. Inflight-v2-data-5b dataset. <https://huggingface.co/datasets/openbmb/Inflight-V2-data-5B>.
- Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, and 1 others. 2024. Tool learning with foundation models. *ACM Computing Surveys*.
- Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. 2020. Compressive transformers for long-range sequence modelling. *Proceedings of ICLR*.
- Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. 2024. Eigen attention: Attention in low-rank space for kv cache compression. *Proceedings of EMNLP findings*.
- Zhenyi Shen, Junru Lu, Lin Gui, Jiazheng Li, Yulan He, Di Yin, and Xing Sun. 2026. Ssa: Sparse sparse attention by aligning full and sparse attention outputs in feature space. In *Proceedings of ICML*.
- Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. *Proceedings of ICML*.
- Jingze Shi, Yifan Wu, Bingheng Wu, Yiran Peng, Liangdong Wang, Guang Liu, and Yuyu Luo. 2025. Trainable dynamic mask sparse attention. *arXiv preprint arXiv:2508.02124*.
- Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.
- Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. 2024. Loki: Low-rank keys for efficient sparse attention. *Proceedings of NeurIPS*.
- Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. Powerinfer: Fast large language model serving with a consumer-grade gpu. *Proceedings of SOSR*.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*.
- Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2025. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *Proceedings of ICML*.
- Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and Jason Wei. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. *Proceedings of ACL Findings*.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. *Proceedings of ICML*.
- Qian Tao, Wenyuan Yu, and Jingren Zhou. 2025. Asymkv: Enabling 1-bit quantization of kv cache with layer-wise asymmetric quantization configurations. *Proceedings of ACL*.

- Pavlo Vasylenko, Hugo Pitorro, André FT Martins, and Marcos Treviso. 2025. Long-context generalization with sparse attention. *arXiv preprint arXiv:2506.16640*.
- Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, and 1 others. 2024a. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Proceedings of NeurIPS*.
- Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, and 1 others. 2024b. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Proceedings of NeurIPS*.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. *Proceedings EMNLP System Demonstrations*.
- Haoyi Wu and Kewei Tu. 2024. Layer-condensed kv cache for efficient inference of large language models. *Proceedings of ACL*.
- Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024a. Infilm: Training-free long-context extrapolation for llms with an efficient context memory. *Proceedings of NeurIPS*.
- Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2025. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *Proceedings of ICLR*.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024b. Efficient streaming language models with attention sinks. *Proceedings of ICLR*.
- Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. 2025. Xattention: Block sparse attention with antidiagonal scoring. *Proceedings of ICML*.
- Ran Yan, Youhe Jiang, Zhuoming Chen, Haohui Mai, Beidi Chen, and Binhang Yuan. 2026. Fsa: An alternative efficient implementation of native sparse attention kernel. *Proceedings of ICLR*.
- Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. 2021. Tuning large neural networks via zero-shot hyperparameter transfer. *Proceedings of NeurIPS*.
- Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. 2025. Lserve: Efficient long-sequence llm serving with unified sparse attention. *Proceedings of MLSys*.
- Shuo Yang, Ying Sheng, Joseph E Gonzalez, Ion Stoica, and Lianmin Zheng. 2024. Post-training sparse attention with double sparsity. *arXiv preprint arXiv:2408.07092*.
- Yao Yao, Zuchao Li, and Hai Zhao. 2024. Sirllm: Streaming infinite retentive llm. *Proceedings of ACL*.
- Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yining Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and 1 others. 2025. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of MLSys*.
- Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. 2025. Helmet: How to evaluate long-context language models effectively and thoroughly. *Proceedings of ICLR*.
- Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, YX Wei, Lean Wang, Zhiping Xiao, and 1 others. 2025. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *Proceedings of ACL*.
- Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*.
- Amir Zandieh, Majid Daliri, and Insu Han. 2025. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead. *Proceedings of AAAI*.
- Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia Wei, Jun Zhu, and Jianfei Chen. 2025a. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization. In *Proceedings of ICML*.
- Jintao Zhang, Jia Wei, Pengle Zhang, Xiaoming Xu, Haofeng Huang, Haoxu Wang, Kai Jiang, Jun Zhu, and Jianfei Chen. 2025b. Sageattention3: Microscaling fp4 attention for inference and an exploration of 8-bit training. *Proceedings of NeurIPS*.
- Jintao Zhang, Jia Wei, Pengle Zhang, Jun Zhu, and Jianfei Chen. 2025c. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *Proceedings of ICLR*.
- Jintao Zhang, Xiaoming Xu, Jia Wei, Haofeng Huang, Pengle Zhang, Chendong Xiang, Jun Zhu, and Jianfei Chen. 2025d. Sageattention2++: A more efficient implementation of sageattention2. *arXiv preprint arXiv:2505.21136*.

- Minwei Zhang, Haifeng Sun, Jingyu Wang, Shaolong Li, Wanyi Ning, Qi Qi, Zirui Zhuang, and Jianxin Liao. 2025e. Clusterattn: Kv cache compression under intrinsic attention clustering. *Proceedings of ACL*.
- Xiangyu Zhang, Yu Zhou, Guang Yang, Harald C Gall, and Taolue Chen. 2025f. Anchor attention, small cache: Code generation with large language models. *IEEE Transactions on Software Engineering*.
- Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. 2023a. Evaluating the performance of large language models on gaokao benchmark. *arXiv preprint arXiv:2305.12474*.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuan-dong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023b. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Proceedings of NeurIPS*.
- Weilin Zhao, Zihan Zhou, Zhou Su, Chaojun Xiao, Yuxuan Li, Yanghao Li, Yudi Zhang, Weilin Zhao, Zhen Li, Yuxiang Huang, and 1 others. 2025. Infillm-v2: Dense-sparse switchable attention for seamless short-to-long adaptation. *arXiv preprint arXiv:2509.24663*.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2024. Sglang: Efficient execution of structured language model programs. *Proceedings of NeurIPS*.
- Qihui Zhou, Peiqi Yin, Pengfei Zuo, and James Cheng. 2025. Sparseserve: Unlocking parallelism for dynamic sparse attention in long-context llm serving. *arXiv preprint arXiv:2509.24626*.

A Table of Content

1	Introduction	1
2	Background	2
2.1	Preliminaries	2
2.2	Observations	3
3	NOSA	3
3.1	Offloadable Trainable Sparse Attention	3
3.2	Ablations on Algorithm Design . . .	5
4	NOSI: Inference System for NOSA	5
5	Experiments	6
5.1	Experimental Setup	7
5.2	Long Context Tasks	7
5.3	General and Reasoning Tasks	8
5.4	Efficiency Benchmarks	8
6	Related Works	8
7	Conclusion	9
A	Table of Content	14

B	Supplementary Experiments	14
B.1	Long-Context Performance	14
B.2	Generalization to Longer Contexts .	15
B.3	Ablation Studies	16
B.3.1	Eviction Head Implementation . .	16
B.3.2	Hyperparameter Analysis	16
B.3.3	Efficiency Under Various Semantics	18
B.3.4	Throughput under Same Batch Size	18
B.3.5	Various Hardware Configurations	18
B.3.6	Wall-Time Breakdown Analysis .	19
C	Reproducibility Settings	20
C.1	Environment	20
C.2	Hyperparameters	20
C.2.1	Model Training	20
C.2.2	Hyperparameters of Each Method	21
C.3	Datasets	22
C.4	Benchmark Settings	23
D	Baselines	24
D.1	Reference and Baseline Methods . .	24
D.2	Other Potentially Related Methods .	25
E	Theoretical Guarantee of Locality	26

B Supplementary Experiments

B.1 Long-Context Performance

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

NOSA achieves the best overall performance among all competing methods. Across the 12 settings in Table 8, 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.

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±0.4	39.0±0.9	60.0±0.7	5.5±0.3	14.3±0.0	17.2±0.3	0.4±0.2	28.3±0.3	
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±1.0	37.5±1.3	55.4±0.5	2.7±0.4	5.8±0.0	16.4±0.3	0.1±0.1	20.6±0.2	
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±0.8	38.4±1.3	59.0±0.4	4.1±0.3	11.0±0.0	16.7±0.3	0.1±0.1	25.5±0.3	
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±1.6	34.3±1.0	57.5±0.3	1.1±1.0	1.8±0.4	16.9±0.7	0.2±0.1	19.8±0.3	
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±1.0	39.6±1.1	54.7±0.4	2.7±0.1	8.7±0.0	15.8±0.7	0.8±0.3	19.7±0.3	
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±1.0	39.8±1.1	59.2±0.6	3.5±0.1	12.7±0.0	17.6±1.0	0.8±0.2	24.0±0.3	
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±1.4	37.2±0.6	58.0±0.1	4.1±0.6	15.5±0.0	17.7±0.3	0.8±0.3	24.4±0.1	
SHADKV ₆₄ ^M	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±0.9	37.9±1.2	50.1±0.3	3.4±0.0	9.6±0.0	17.3±0.5	0.2±0.1	20.7±0.2	
SHADKV ₈ ^M	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±0.6	38.5±0.9	53.9±0.4	4.2±0.2	11.2±0.0	17.5±0.5	0.1±0.1	23.9±0.2	
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±0.3	34.4±0.3	48.8±0.5	1.2±0.3	2.6±0.0	13.2±0.3	0.2±0.1	15.0±0.1	
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±0.3	36.4±0.4	51.7±0.3	1.1±0.2	2.0±0.0	13.0±0.6	0.4±0.2	15.5±0.2	
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±0.8	28.2±0.7	48.7±0.3	2.8±0.3	7.6±0.0	14.6±0.7	0.4±0.1	16.7±0.1	
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±1.0	39.6±0.3	57.5±0.4	3.6±0.4	13.6±0.0	16.2±0.7	0.6±0.1	22.9±0.2	
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±1.1	50.6±0.2	49.5±0.5	9.0±0.2	23.1±0.0	21.5±0.6	5.0±0.2	29.6±0.3	
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±1.2	48.3±0.6	46.4±0.6	4.4±0.4	11.6±0.0	20.6±0.4	6.2±0.4	22.9±0.3	
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±0.7	49.0±0.3	48.0±0.9	5.5±0.4	15.0±0.0	21.2±0.8	7.3±0.2	26.5±0.3	
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±0.9	50.4±0.3	49.3±0.6	5.9±0.5	21.1±0.6	22.1±0.1	5.7±0.6	26.1±0.2	
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±0.1	47.2±0.7	47.6±1.2	2.8±0.3	16.0±0.0	23.9±0.7	4.4±0.1	22.1±0.3	
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±0.3	49.8±0.3	47.7±0.3	6.3±0.4	24.3±0.0	21.5±0.8	2.7±0.3	28.2±0.1	
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±0.6	49.4±0.4	48.6±0.9	5.7±1.3	22.7±0.0	23.0±1.1	3.2±0.5	27.5±0.5	
SHADKV ₆₄ ^M	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±0.6	48.0±1.1	39.1±0.7	5.0±0.9	13.5±0.2	21.2±1.1	5.6±0.4	22.8±0.6	
SHADKV ₈ ^M	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±0.5	48.9±1.0	40.5±0.6	7.4±0.7	17.1±0.0	21.1±1.0	5.6±0.7	26.3±0.3	
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±1.4	44.2±0.4	35.6±0.5	1.7±0.4	2.8±0.0	12.7±1.8	5.6±0.4	15.4±0.3	
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±0.4	44.1±0.5	37.5±0.2	1.5±0.5	1.9±0.0	12.0±1.9	4.9±0.6	15.2±0.2	
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±0.6	41.2±0.9	41.1±0.0	3.6±0.4	10.1±0.0	21.5±0.8	5.3±0.6	19.3±0.2	
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±1.0	49.2±0.6	52.2±0.4	6.4±0.2	26.4±0.0	21.6±0.7	2.6±1.1	27.8±0.3	
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±0.3	56.1±1.0	61.3±0.4	14.5±0.4	39.8±0.0	26.6±0.6	10.7±1.0	42.6±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±0.2	55.6±0.9	58.9±0.6	10.7±0.6	21.3±0.1	25.2±1.2	7.8±0.5	33.5±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±0.4	55.7±1.1	60.6±0.5	12.4±0.9	29.3±0.0	26.6±0.8	8.8±1.3	38.8±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±1.5	55.6±0.8	61.5±0.6	10.7±2.4	35.1±0.1	26.5±1.5	8.2±0.8	36.2±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±0.2	52.4±0.6	63.8±0.1	10.4±0.7	26.5±0.0	25.7±1.9	6.5±0.7	31.2±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	86.3±0.0	56.3±0.4	60.6±0.5	13.6±1.1	30.8±0.0	26.7±1.5	8.5±0.7	40.4±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±0.2	54.1±1.1	68.9±0.3	13.8±0.7	37.5±0.0	26.6±1.5	8.6±0.6	40.3±0.3	
SHADKV ₆₄ ^M	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±0.9	53.1±0.7	52.4±0.5	10.7±1.1	23.4±0.0	23.9±1.1	8.1±0.6	32.3±0.4	
SHADKV ₈ ^M	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±0.2	53.1±0.7	53.5±0.2	11.6±0.5	28.9±0.0	25.9±2.1	7.7±1.0	36.5±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±0.1	47.2±0.5	36.7±0.9	1.7±0.1	2.2±0.0	16.6±2.5	8.1±0.4	16.9±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±0.5	47.8±1.2	37.4±0.8	1.4±0.5	2.4±0.0	13.5±3.9	7.0±1.2	16.4±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±0.7	39.9±0.7	56.1±0.4	9.3±0.9	21.6±0.0	28.2±1.8	6.5±0.8	26.3±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±0.8	54.3±1.2	65.1±0.5	12.5±1.3	31.0±0.0	25.1±1.8	8.2±0.7	37.6±0.7	

Table 8: *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.

B.2 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× and 4× length generalization beyond the training length. For the 32K setting, we use the same protocol as in Sec. 5.2. 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 9, from which we draw the following conclusion.

NOSA also shows superior performance in length extrapolation. As shown in Table 9, 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.

Tasks	Full Prefill						Sparse Prefill						
	FULLATTN	SHADKV ₆₄	SHADKV ₈	ARKVALE	DMA ^F	NOSA ^F	INFLLMv2	SHADKV ₆₄ ^M	SHADKV ₈ ^M	INFLLM ₁₂₈	INFLLM ₆₄	DMA ^S	NOSA ^S
Input Length: 32K													
Recall	56.3 $\pm$ 1.8	31.0 $\pm$ 0.6	48.8 $\pm$ 3.4	41.6 $\pm$ 2.5	10.0 $\pm$ 1.8	56.6 $\pm$ 4.2	45.3 $\pm$ 2.0	30.4 $\pm$ 2.7	42.6 $\pm$ 3.2	15.0 $\pm$ 3.2	15.6 $\pm$ 1.9	8.4 $\pm$ 2.7	35.0 $\pm$ 2.5
RAG	50.7 $\pm$ 0.5	49.1 $\pm$ 1.3	49.3 $\pm$ 2.0	50.2 $\pm$ 1.5	47.9 $\pm$ 2.5	51.4 $\pm$ 1.4	49.7 $\pm$ 2.2	44.5 $\pm$ 0.9	46.4 $\pm$ 1.6	44.9 $\pm$ 2.9	45.1 $\pm$ 3.1	32.2 $\pm$ 3.9	44.7 $\pm$ 3.2
ICL	67.3 $\pm$ 2.6	66.7 $\pm$ 5.7	66.7 $\pm$ 2.5	65.7 $\pm$ 4.9	59.3 $\pm$ 4.0	64.7 $\pm$ 3.8	69.7 $\pm$ 2.5	60.0 $\pm$ 1.7	64.0 $\pm$ 1.4	38.3 $\pm$ 2.6	38.3 $\pm$ 1.2	53.3 $\pm$ 3.3	68.7 $\pm$ 5.4
Cite	13.9 $\pm$ 3.7	9.9 $\pm$ 2.3	10.7 $\pm$ 0.6	9.7 $\pm$ 1.5	10.3 $\pm$ 0.2	12.7 $\pm$ 1.1	12.6 $\pm$ 3.7	9.8 $\pm$ 1.2	9.2 $\pm$ 1.9	13.0 $\pm$ 1.7	10.9 $\pm$ 1.4	9.5 $\pm$ 1.3	10.0 $\pm$ 1.9
Rerank	30.8 $\pm$ 0.5	9.1 $\pm$ 0.5	19.6 $\pm$ 1.9	19.4 $\pm$ 1.9	13.3 $\pm$ 1.0	22.0 $\pm$ 4.0	20.0 $\pm$ 3.3	6.6 $\pm$ 1.8	13.8 $\pm$ 2.2	4.3 $\pm$ 1.0	4.8 $\pm$ 1.7	3.2 $\pm$ 0.8	16.4 $\pm$ 1.6
LongQA	38.0 $\pm$ 7.0	34.7 $\pm$ 4.7	36.3 $\pm$ 5.6	32.8 $\pm$ 3.2	34.8 $\pm$ 5.7	34.2 $\pm$ 6.8	32.5 $\pm$ 4.7	33.2 $\pm$ 3.2	32.3 $\pm$ 2.6	10.6 $\pm$ 1.4	10.9 $\pm$ 3.4	33.9 $\pm$ 1.4	30.6 $\pm$ 6.0
Summ.	10.1 $\pm$ 2.7	10.9 $\pm$ 2.2	11.2 $\pm$ 3.0	8.2 $\pm$ 2.0	4.5 $\pm$ 1.2	10.6 $\pm$ 0.9	8.3 $\pm$ 2.1	8.0 $\pm$ 0.7	5.7 $\pm$ 1.3	9.0 $\pm$ 0.8	7.9 $\pm$ 2.4	7.3 $\pm$ 2.9	9.6 $\pm$ 3.4
Avg. $\uparrow$	38.2 $\pm$ 1.3	30.2 $\pm$ 1.4	34.6 $\pm$ 1.6	32.5 $\pm$ 1.1	25.7 $\pm$ 1.5	36.0 $\pm$ 1.5	34.0 $\pm$ 1.9	27.5 $\pm$ 0.7	30.6 $\pm$ 0.6	19.3 $\pm$ 1.2	19.1 $\pm$ 0.9	21.1 $\pm$ 0.6	30.7 $\pm$ 2.2
Input Length: 64K													
Recall	38.2 $\pm$ 2.8	20.5 $\pm$ 0.8	26.3 $\pm$ 2.6	23.0 $\pm$ 2.9	5.4 $\pm$ 0.9	40.4 $\pm$ 1.7	30.2 $\pm$ 2.6	16.6 $\pm$ 1.4	29.0 $\pm$ 4.6	12.2 $\pm$ 1.9	12.2 $\pm$ 2.0	6.8 $\pm$ 2.0	27.0 $\pm$ 1.5
RAG	44.9 $\pm$ 2.6	44.8 $\pm$ 1.9	44.7 $\pm$ 3.2	45.3 $\pm$ 2.2	40.5 $\pm$ 3.5	45.0 $\pm$ 3.5	42.4 $\pm$ 1.1	38.6 $\pm$ 1.4	38.2 $\pm$ 1.9	42.6 $\pm$ 3.6	41.7 $\pm$ 3.2	32.9 $\pm$ 5.3	42.5 $\pm$ 1.5
ICL	58.0 $\pm$ 3.7	56.7 $\pm$ 3.7	59.3 $\pm$ 2.9	61.3 $\pm$ 3.7	59.7 $\pm$ 1.2	49.3 $\pm$ 1.2	70.3 $\pm$ 4.9	59.3 $\pm$ 3.4	59.7 $\pm$ 2.6	44.0 $\pm$ 5.0	40.0 $\pm$ 2.2	58.0 $\pm$ 0.8	71.3 $\pm$ 6.1
Cite	12.9 $\pm$ 1.8	9.0 $\pm$ 1.5	10.0 $\pm$ 2.0	12.1 $\pm$ 0.7	11.0 $\pm$ 1.2	10.0 $\pm$ 1.3	8.8 $\pm$ 2.0	7.0 $\pm$ 1.1	8.3 $\pm$ 1.8	12.1 $\pm$ 1.5	11.5 $\pm$ 2.5	7.0 $\pm$ 1.6	7.6 $\pm$ 0.9
Rerank	11.2 $\pm$ 1.7	0.5 $\pm$ 0.2	2.8 $\pm$ 1.1	9.5 $\pm$ 1.2	7.4 $\pm$ 0.4	8.6 $\pm$ 0.7	6.8 $\pm$ 2.6	0.8 $\pm$ 0.8	0.8 $\pm$ 0.7	1.8 $\pm$ 0.3	1.7 $\pm$ 0.4	3.3 $\pm$ 0.6	5.3 $\pm$ 0.6
LongQA	38.1 $\pm$ 6.7	36.8 $\pm$ 3.7	36.9 $\pm$ 4.6	36.1 $\pm$ 6.7	27.2 $\pm$ 2.0	36.3 $\pm$ 3.0	37.0 $\pm$ 4.1	34.3 $\pm$ 2.2	34.6 $\pm$ 2.2	14.8 $\pm$ 2.5	16.9 $\pm$ 6.8	29.9 $\pm$ 4.9	35.9 $\pm$ 2.4
Summ.	8.2 $\pm$ 1.8	6.7 $\pm$ 2.2	5.6 $\pm$ 1.0	6.2 $\pm$ 2.5	3.5 $\pm$ 0.1	4.6 $\pm$ 1.4	6.3 $\pm$ 1.3	2.7 $\pm$ 0.6	3.0 $\pm$ 0.2	11.1 $\pm$ 3.0	10.4 $\pm$ 0.4	3.9 $\pm$ 1.3	5.1 $\pm$ 2.8
Avg. $\uparrow$	30.2 $\pm$ 1.9	25.0 $\pm$ 1.7	26.5 $\pm$ 1.8	27.6 $\pm$ 2.2	22.1 $\pm$ 0.9	27.7 $\pm$ 1.3	28.8 $\pm$ 0.8	22.8 $\pm$ 0.5	24.7 $\pm$ 0.8	19.8 $\pm$ 1.5	19.2 $\pm$ 1.8	20.3 $\pm$ 1.2	27.8 $\pm$ 1.3

Table 9: **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.

B.3 Ablation Studies

B.3.1 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 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 11.

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 12, 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.

B.3.2 Hyperparameter Analysis

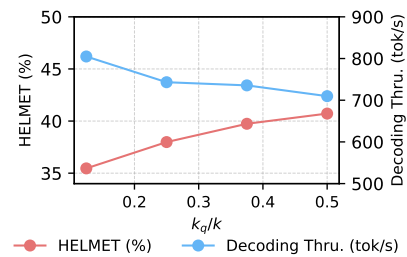


Figure 9: HELMET score and decoding throughput under various k_q/k .

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

Tasks	Full Prefill						Sparse Prefill							
	FULLATTN	SHADKV ₆₄	SHADKV ₈	ARKVALE	DMA ^F	NOSA ^F	INFLLMv2	SHADKV ₆₄ ^M	SHADKV ₈ ^M	INFLLM ₁₂₈	INFLLM ₆₄	DMA ^S	NOSA ^S	
Budget $k = 2048$ Sparsity: 0.125														
Recall	88.0 $\pm$ 0.8	40.3 $\pm$ 4.3	72.6 $\pm$ 1.1	49.3 $\pm$ 3.8	17.5 $\pm$ 1.1	73.9 $\pm$ 0.8	54.2 $\pm$ 0.9	30.5 $\pm$ 3.0	56.0 $\pm$ 2.8	20.3 $\pm$ 2.3	20.4 $\pm$ 2.2	10.7 $\pm$ 2.1	40.4 $\pm$ 2.5	
RAG	54.5 $\pm$ 1.9	53.3 $\pm$ 2.5	53.8 $\pm$ 2.2	54.0 $\pm$ 2.6	52.2 $\pm$ 3.2	56.4 $\pm$ 2.0	51.8 $\pm$ 2.3	44.5 $\pm$ 2.9	43.6 $\pm$ 1.4	46.3 $\pm$ 3.6	46.7 $\pm$ 2.2	28.7 $\pm$ 1.8	46.8 $\pm$ 1.4	
ICL	59.7 $\pm$ 2.1	55.7 $\pm$ 0.5	57.3 $\pm$ 2.1	60.7 $\pm$ 2.6	63.7 $\pm$ 2.5	59.0 $\pm$ 2.4	66.3 $\pm$ 3.3	52.3 $\pm$ 2.6	49.3 $\pm$ 5.3	41.3 $\pm$ 3.4	42.7 $\pm$ 2.1	42.3 $\pm$ 3.9	60.7 $\pm$ 5.4	
Cite	13.9 $\pm$ 3.7	9.7 $\pm$ 0.6	8.6 $\pm$ 0.5	10.9 $\pm$ 1.1	9.2 $\pm$ 2.8	11.9 $\pm$ 2.3	12.7 $\pm$ 0.9	4.9 $\pm$ 0.9	6.9 $\pm$ 1.5	8.3 $\pm$ 0.8	9.6 $\pm$ 1.2	7.8 $\pm$ 2.9	12.7 $\pm$ 1.0	
Rerank	36.4 $\pm$ 3.1	13.3 $\pm$ 1.6	26.5 $\pm$ 2.3	29.2 $\pm$ 2.5	17.4 $\pm$ 2.2	24.7 $\pm$ 2.6	26.4 $\pm$ 1.7	6.0 $\pm$ 0.7	13.1 $\pm$ 1.2	8.1 $\pm$ 0.5	8.2 $\pm$ 0.9	8.0 $\pm$ 1.6	22.6 $\pm$ 1.8	
LongQA	31.0 $\pm$ 2.2	27.9 $\pm$ 4.2	32.6 $\pm$ 4.3	28.8 $\pm$ 1.2	29.5 $\pm$ 3.4	31.2 $\pm$ 1.3	28.5 $\pm$ 3.6	25.0 $\pm$ 2.3	25.5 $\pm$ 2.9	9.2 $\pm$ 4.0	12.5 $\pm$ 4.2	29.6 $\pm$ 2.6	28.6 $\pm$ 4.6	
Summ.	11.3 $\pm$ 4.8	9.8 $\pm$ 4.8	7.0 $\pm$ 3.2	7.6 $\pm$ 2.0	4.2 $\pm$ 0.6	7.4 $\pm$ 3.7	9.1 $\pm$ 3.2	4.1 $\pm$ 0.8	3.9 $\pm$ 2.5	8.2 $\pm$ 3.8	9.1 $\pm$ 4.5	6.7 $\pm$ 2.7	7.0 $\pm$ 1.3	
Avg. $\uparrow$	42.1 $\pm$ 0.8	30.0 $\pm$ 1.2	36.9 $\pm$ 1.2	34.3 $\pm$ 0.9	27.7 $\pm$ 0.2	37.8 $\pm$ 0.4	35.6 $\pm$ 0.8	23.9 $\pm$ 0.5	28.3 $\pm$ 1.1	20.2 $\pm$ 0.7	21.3 $\pm$ 0.4	19.1 $\pm$ 1.0	31.3 $\pm$ 1.9	
Budget $k = 4096$ Sparsity: 0.250														
Recall	88.0 $\pm$ 0.8	56.3 $\pm$ 3.6	78.5 $\pm$ 1.1	58.6 $\pm$ 1.6	31.9 $\pm$ 4.9	85.5 $\pm$ 1.2	74.2 $\pm$ 3.6	55.7 $\pm$ 6.2	70.5 $\pm$ 3.0	26.8 $\pm$ 1.3	32.4 $\pm$ 2.7	20.9 $\pm$ 1.0	67.2 $\pm$ 1.3	
RAG	54.5 $\pm$ 1.9	53.8 $\pm$ 1.7	53.1 $\pm$ 2.2	54.3 $\pm$ 2.1	53.1 $\pm$ 2.8	56.2 $\pm$ 3.0	53.1 $\pm$ 2.0	53.4 $\pm$ 1.8	51.8 $\pm$ 1.6	46.3 $\pm$ 2.1	46.4 $\pm$ 2.4	40.0 $\pm$ 2.5	52.6 $\pm$ 1.7	
ICL	59.7 $\pm$ 2.1	57.3 $\pm$ 3.9	58.3 $\pm$ 0.5	61.3 $\pm$ 2.6	67.3 $\pm$ 0.9	61.3 $\pm$ 1.7	72.0 $\pm$ 0.8	53.7 $\pm$ 3.4	55.0 $\pm$ 1.4	39.0 $\pm$ 2.2	38.0 $\pm$ 3.7	59.3 $\pm$ 1.2	67.0 $\pm$ 2.2	
Cite	13.9 $\pm$ 3.7	9.6 $\pm$ 2.6	8.9 $\pm$ 2.3	11.6 $\pm$ 1.3	9.9 $\pm$ 0.5	12.4 $\pm$ 0.5	12.8 $\pm$ 3.5	9.7 $\pm$ 1.2	7.9 $\pm$ 2.3	12.8 $\pm$ 1.4	11.2 $\pm$ 1.6	9.6 $\pm$ 1.5	10.4 $\pm$ 0.4	
Rerank	36.4 $\pm$ 3.1	23.7 $\pm$ 1.7	28.6 $\pm$ 2.3	32.8 $\pm$ 2.6	25.8 $\pm$ 3.8	33.4 $\pm$ 0.9	35.5 $\pm$ 2.9	20.8 $\pm$ 3.5	25.5 $\pm$ 3.6	12.4 $\pm$ 0.5	14.1 $\pm$ 0.2	17.3 $\pm$ 1.8	30.4 $\pm$ 2.7	
LongQA	31.0 $\pm$ 2.2	29.2 $\pm$ 4.4	31.9 $\pm$ 6.3	30.8 $\pm$ 1.9	30.7 $\pm$ 2.5	31.4 $\pm$ 1.7	31.4 $\pm$ 1.6	26.4 $\pm$ 3.4	31.9 $\pm$ 7.0	11.6 $\pm$ 3.5	11.1 $\pm$ 1.9	31.5 $\pm$ 4.6	28.5 $\pm$ 3.6	
Summ.	11.3 $\pm$ 4.8	9.9 $\pm$ 4.2	10.4 $\pm$ 3.4	8.9 $\pm$ 3.5	8.3 $\pm$ 2.6	9.1 $\pm$ 2.4	11.2 $\pm$ 3.6	11.1 $\pm$ 3.0	8.7 $\pm$ 3.2	9.1 $\pm$ 2.5	10.0 $\pm$ 5.8	9.4 $\pm$ 4.3	8.9 $\pm$ 2.3	
Avg. $\uparrow$	42.1 $\pm$ 0.8	34.3 $\pm$ 0.5	38.5 $\pm$ 1.7	36.9 $\pm$ 0.5	32.4 $\pm$ 0.6	41.3 $\pm$ 0.4	41.5 $\pm$ 0.8	33.0 $\pm$ 1.1	35.9 $\pm$ 0.1	22.6 $\pm$ 0.4	23.3 $\pm$ 1.6	26.9 $\pm$ 0.6	37.9 $\pm$ 0.8	
Budget $k = 6144$ Sparsity: 0.375														
Recall	88.0 $\pm$ 0.8	68.0 $\pm$ 2.0	79.7 $\pm$ 1.6	68.0 $\pm$ 3.0	46.2 $\pm$ 2.9	93.6 $\pm$ 2.2	86.7 $\pm$ 1.6	61.0 $\pm$ 4.9	71.4 $\pm$ 3.4	34.4 $\pm$ 1.4	38.1 $\pm$ 3.1	36.1 $\pm$ 2.1	83.1 $\pm$ 4.6	
RAG	54.5 $\pm$ 1.9	54.0 $\pm$ 2.0	53.0 $\pm$ 2.6	55.1 $\pm$ 2.4	53.5 $\pm$ 3.0	56.4 $\pm$ 1.7	55.1 $\pm$ 2.0	53.3 $\pm$ 2.1	51.9 $\pm$ 1.3	46.3 $\pm$ 1.7	46.3 $\pm$ 2.9	47.6 $\pm$ 1.4	53.6 $\pm$ 0.9	
ICL	59.7 $\pm$ 2.1	58.3 $\pm$ 2.4	58.7 $\pm$ 0.9	61.0 $\pm$ 2.2	67.3 $\pm$ 0.5	63.0 $\pm$ 2.2	68.0 $\pm$ 1.4	55.0 $\pm$ 2.2	55.3 $\pm$ 1.9	40.3 $\pm$ 2.1	41.3 $\pm$ 0.5	65.3 $\pm$ 0.9	66.7 $\pm$ 2.6	
Cite	13.9 $\pm$ 3.7	10.6 $\pm$ 0.9	10.5 $\pm$ 1.2	10.5 $\pm$ 0.9	10.7 $\pm$ 1.3	10.7 $\pm$ 1.3	14.5 $\pm$ 4.4	8.8 $\pm$ 1.4	8.2 $\pm$ 1.7	12.1 $\pm$ 1.5	11.4 $\pm$ 1.9	10.6 $\pm$ 1.1	12.0 $\pm$ 1.5	
Rerank	36.4 $\pm$ 3.1	25.4 $\pm$ 2.1	30.1 $\pm$ 2.2	33.5 $\pm$ 2.1	30.4 $\pm$ 1.6	37.0 $\pm$ 2.6	39.7 $\pm$ 1.0	25.0 $\pm$ 1.8	26.4 $\pm$ 4.0	18.2 $\pm$ 1.7	20.5 $\pm$ 0.3	26.3 $\pm$ 2.0	36.9 $\pm$ 1.3	
LongQA	31.0 $\pm$ 2.2	29.7 $\pm$ 4.2	31.2 $\pm$ 6.2	30.5 $\pm$ 0.8	32.5 $\pm$ 3.6	31.8 $\pm$ 0.9	30.2 $\pm$ 1.8	26.1 $\pm$ 4.3	32.1 $\pm$ 7.7	11.4 $\pm$ 2.3	10.6 $\pm$ 1.4	31.1 $\pm$ 5.5	31.3 $\pm$ 2.4	
Summ.	11.3 $\pm$ 4.8	9.0 $\pm$ 4.0	7.8 $\pm$ 2.5	10.0 $\pm$ 2.8	7.7 $\pm$ 1.0	10.2 $\pm$ 2.7	9.9 $\pm$ 4.7	9.9 $\pm$ 4.4	7.2 $\pm$ 2.7	10.0 $\pm$ 2.0	9.8 $\pm$ 4.9	10.8 $\pm$ 2.9	11.5 $\pm$ 1.9	
Avg. $\uparrow$	42.1 $\pm$ 0.8	36.4 $\pm$ 0.5	38.7 $\pm$ 1.5	38.4 $\pm$ 0.8	35.5 $\pm$ 0.6	43.3 $\pm$ 0.5	43.4 $\pm$ 0.8	34.2 $\pm$ 0.3	36.1 $\pm$ 0.1	24.7 $\pm$ 0.4	25.5 $\pm$ 0.7	32.6 $\pm$ 1.1	42.1 $\pm$ 1.0	

Table 10: **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.

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 11: Details of various implementations of the eviction head.

NOSA on the 8B size model on HELMET, following App. C’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, 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 $k \in \{4096, 6144\}$

Implementation	SG1	SG2	SG3	MK1	MK2	MK3	MV	MQ	VT	CWE	FWE	QA1	QA2	Avg. $\uparrow$
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

Table 12: RULER evaluation results with various implementations of the eviction head.

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 App. C. For each task, we sample 20 instances and report the results in Table 10.

A larger budget improves long-context performance. As shown in Table 10, 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.

B.3.3 Efficiency Under Various Semantics

Task	T1.1	T1.2	T2.1	T2.2	T3.1	T3.2
Ctx. Req.	Full	Partial	Full	Partial	Full	Partial
INFLLMv2+NOSI, Offload	561.0	576.7	590.0	563.5	568.4	550.1
NOSA+NOSI, No Offload	220.8	220.3	215.5	215.4	218.6	211.4
NOSA+NOSI, Offload	700.4	712.5	714.4	702.6	709.9	695.9

Table 13: Decoding throughput (tok/s) $\uparrow$. EB: 16, input length: 16K. “Ctx. Req.” refer to *LongBench-Pro*’s context requirement.

Since NOSA relies on semantic locality to achieve higher decoding throughput, the degree of locality may vary across input contexts. To demonstrate that NOSA can consistently improve decoding throughput, we benchmark NOSA with offloading on NOSI against the strongest decoding-

throughput baselines (INFLLMv2+NOSI with offloading, and NOSA+NOSI without offloading) on LongBench-Pro (Chen et al., 2026). By design, LongBench-Pro categorizes inputs into full-context and partial-context settings, which exhibit different levels of locality. As shown in Table 13, NOSA+NOSI outperforms the baselines under diverse context requirements. Its decoding throughput remains stable in both full-context and partial-context settings, demonstrating the generalizability of the proposed method, NOSA.

B.3.4 Throughput under Same Batch Size

To demonstrate the efficiency of NOSI, we measure the decoding throughput at the same batch size. Specifically, we use $B = 16$ at 16K input length.

As shown in Table 14, when the batch size is fixed, offloading can reduce generation throughput because PCIe transfer introduces additional communication cost. Importantly, NOSA suffers a smaller throughput drop than INFLLMv2 after offloading (-9.1% v.s. -31.6%), which supports our claim that NOSA’s stronger locality reduces CPU-GPU communication.

B.3.5 Various Hardware Configurations

Various NUMA configurations. We benchmark NOSA+NOSI under different NUMA settings by varying the number of cores, memory and core assignments, and the number of GPUs, as shown in Table 15. The results show that NUMA configuration has little effect on the efficiency of NOSA+NOSI. When reducing the number of cores per GPU, we observe only minor degradation in decoding throughput, and 24/26 cores already achieve strong throughput. Notably, this configuration can be easily supported by a 192-core or 208-core machine, showing that NOSA+NOSI remains effective under various hardware configurations.

Efficiency on Improved Hardware. To investigate whether NOSA+NOSI can maintain strong efficiency on more advanced hardware, we benchmark the proposed method on NVIDIA H100

Method	FULLATTN+HF	FULLATTN+vLLM	FULLATTN+SGLANG	INFLLMv2+HF	INFLLMv2+NOSI	INFLLMv2+NOSI+OFFLOAD
Thru (tok/s)	OOM	678.41	774.74	OOM	824.09	563.16
Thru (tok/s)	SHADOWKV	ARKVALE	INFLLM	NOSA+HF	NOSA+NOSI	NOSA+NOSI+OFFLOAD
Method	585.16	–	35.00	OOM	817.16	743.18

Table 14: Throughput under same batch size.

#Core	NUMA Mem.	Index Core	Thru.	Comment
26	0	0	312.39	Main experiment.
26	0	1	308.49	Cross NUMA (complete).
26	0	Half 0, half 1	309.22	Cross NUMA (half).
13	0	0	295.86	104 cores, 8 GPUs.
24	0	0	308.13	192 cores, 8 GPUs.

Table 15: Decoding throughput (tok/s) $\uparrow$ under different NUMA setting. EB: 16, input length: 16K.

Input Length	INFLLMv2 + NOSI Offload	NOSA+NOSI No Offload	NOSA+NOSI Offload
16K	1762.16	219.38	2551.66
32K	1535.70	436.45	2024.04
64K	1510.80	OOM	2034.64

Table 16: Decoding throughput (tok/s) $\uparrow$ on NVIDIA H100 and PCIe 5.0. Effective batch size: 16.

GPUs with PCIe 5.0. As shown in Table 16, all methods achieve higher decoding efficiency due to the increased bandwidth of both HBM and the PCIe bus. Nevertheless, NOSA+NOSI still outperforms the baselines and achieves significantly higher throughput.

More Discussion. Communication media can also be NVLink (which is significantly faster than IB), or disk bus (significantly slower). Here, we would like to provide a short discussion on these scenarios. For NVLink, it is rare to have a cluster where the GPU and CPU are connected via NVLink. For GPU-GPU connections via NVLink, the other GPU is commonly used as a compute node rather than a KV-cache buffer. Therefore, we believe factoring out NVLink as the GPU-buffer transmitter is a reasonable choice. For slower memory systems, such as the I/O bus of a disk, NOSA withNOSI can potentially work on these scenarios due to enhanced locality. Due to resource limit, we would not be able to test them now, and we leave them to future work.

B.3.6 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.

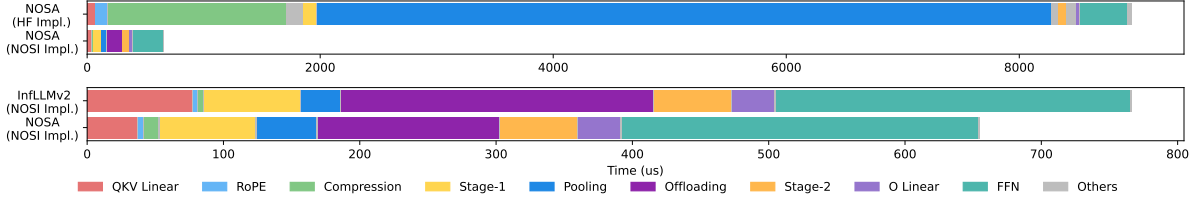


Figure 10: Decoding wall-time breakdown analysis.

C Reproducibility Settings

C.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.

C.2 Hyperparameters

C.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 17. 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 and 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).

Fairness of Competition with Baselines. Here, we would like to clarify the fairness of the competition between our method with training-free offloading baselines. The training-free offloading baselines in our paper are not applied directly to the original short-context base model; they are applied after the same long-context CPT stage, because without long-context adaptation these models cannot reliably handle long contexts. Therefore, all compared methods already rely on long-context CPT. NOSA does not add another round of training, nor does it use additional long-context data. It simply replaces the dense-attention CPT used before post-training offloading with an offloading-aware sparse-attention CPT, so that the model is trained under the locality constraint that will be used at inference time. This design is motivated by the following observations:

(I) Long-context CPT is a standard stage for modern LLMs when extending their context length. Whether one later applies offloading in a training-free manner or trains a sparse/offloaded attention mechanism, this CPT stage is still needed. Therefore, NOSA does not add training cost on top of the long-context adaptation pipeline; it changes the attention mechanism used during that stage.

(II) Recent trainable sparse attention methods, such as NSA and InfLLMv2, show that training with sparse attention during CPT (sparse→CPT) generally outperforms applying sparsity only after dense CPT (CPT→sparse). NOSA follows the same motivation, but further incorporates the locality requirement imposed by KV offloading.

(III) Offloading introduces a locality constraint that is absent from ordinary sparse attention. We therefore compare three settings: (a) post-training offloading, where CPT is conducted with dense attention and offloading is applied only afterward; (b) directly offloading a trainable sparse attention model, where CPT uses sparse attention but without explicit locality constraints; and (c) NOSA,

where CPT uses sparse attention under offloading-aware locality constraints. Empirically, NOSA provides the best tradeoff between performance and decoding efficiency.

In our experiments, baselines such as SHADOWKV, INFLLM, and ARKVALE are also applied after long-context CPT with full attention, since the base models are initially pretrained with shorter context lengths. Thus, NOSA should be understood as changing the timing and constraints of sparse/offloaded attention within the standard CPT process, rather than adding a new training stage.

Potential Risks of Model and Dataset Artifacts.

Both MiniCPM4 models and the InfLLMv2-data-5B dataset are released under the Apache-2.0 license, and we comply with the license requirements throughout our research. We use these artifacts for research purposes only, which is consistent with their intended use. We acknowledge that large-scale training datasets may inadvertently contain personally identifiable information or offensive content, and InfLLMv2-data-5B may share such risks. In this work, aligned with common practice in related long-context pretraining studies, we directly use the publicly released long-context pretraining dataset without additional preprocessing. Since we do not perform further data collection or modification, we rely on the dataset provider’s preprocessing procedures, including anonymization and safety filtering, to mitigate these risks. The MiniCPM4 models are multilingual LLMs. They are designed for natural language generation and cover a wide range of domains represented in their pretraining data. Similarly, InfLLMv2-data-5B is a multilingual dataset covering multiple domains, primarily intended for long-context modeling. In total, we use approximately 5B tokens from this dataset, as shown in Table 17. All entries are used for training. No validation is conducted during training, and model performance is evaluated only after training on downstream benchmarks.

C.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 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): SHADKV₈, SHADKV₆₄, SHADKV₈^M, and SHADKV₆₄^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 SHADKV₈ and SHADKV₈^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 SHADKV₆₄ and SHADKV₆₄^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 SHADKV₈ 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

Model Size	1B	3B	8B
Architecture	Llama-3	Llama-3	MiniCPM4
Use μ P (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 Dimension	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

Table 17: Model architecture and training details.

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, INFLLM₆₄ and INFLLM₁₂₈, 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.

C.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 3 and Table 5. 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.

C.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.

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

A ship traveling along a river has covered $\$24 \backslash \mathrm{~km}$ upstream and $\$28 \backslash \mathrm{~km}$ downstream. For this journey, it took half an hour less than for traveling $\$30 \backslash \mathrm{~km}$ upstream and $\$21 \backslash \mathrm{~km}$ downstream, or half an hour more than for traveling $\$15 \backslash \mathrm{~km}$ upstream and $\$42 \backslash \mathrm{~km}$ 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 $\backslash \backslash (v \backslash \backslash)$ be the speed of the ship in still water (in km/h) and $\backslash \backslash (r \backslash \backslash)$ be the speed

...

The speed of the ship in still water is $\backslash \backslash (\backslash \boxed{10} \backslash \backslash)$ 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  $\sum_{n=1}^{\infty} \frac{1}{2^n}$  of
...
Thus, the least possible number  $\sum_{n=1}^{\infty} \frac{1}{2^n}$  is
 $\sum_{n=1}^{\infty} \frac{1}{2^n}$ .

```

Question: {Input Question} [Real input question]

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

```

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}

```

Listing 2: LLM-judge prompt.

Efficiency Benchmark. All results in Table 7 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

$$EB_x = \frac{BM_x}{M_{\text{NOSA}}}. \quad (8)$$

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

$$B_x = EB \cdot \frac{M_{\text{NOSA}}}{M_x}. \quad (9)$$

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 across 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.

D Baselines

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

D.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 SEER-ATTENTION (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.

D.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 PRU-LONG (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 and 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., 2025f), 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.**

E Theoretical Guarantee of Locality

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

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

$$\Gamma_q(t-1) = \{a_1, a_2, \dots, a_{k_q}\}, \quad (10)$$

$$\text{s.t. } s_{a_1}^q \geq s_{a_2}^q \geq \dots \geq s_{a_{k_q}}^q. \quad (11)$$

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

$$\text{ArgTop}_k(\{s_i^e\}_{i=1}^n) = \{b_1, b_2, \dots, b_k\}, \quad (12)$$

$$\text{s.t. } s_{b_1}^e \geq s_{b_2}^e \geq \dots \geq s_{b_k}^e. \quad (13)$$

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

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

Therefore, we have

$$b_1, b_2, \dots, b_{k-p} \in \Gamma(t-1). \quad (14)$$

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