Optimizing Native Sparse Attention with Latent Attention and Local Global Alternating Strategies

Yuxuan Hu[1,2,4], **Jianchao Tan**[4], **Jiaqi Zhang**[4], **Wen Zan**[4], **Pingwei Sun**[4]
Yifan Lu[4], **Xunliang Cai**[4], **Jing Zhang**[1,3*]
[1]School of Information, Renmin University of China, Beijing, China
[2]Key Laboratory of Data Engineering and Knowledge Engineering, Beijing, China
[3]Engineering Research Center of Database and Business Intelligence, MOE, China
[4]Meituan, Beijing, China
{huyuxuan1999,zhang-jing}@ruc.edu.cn

Abstract

In this work, we conduct a systematic analysis of Native Sparse Attention (NSA) and propose targeted improvements that enhance long-context modeling. A key insight is that alternating between local (sliding-window) and global (compression/selective) attention across layers, rather than using fixed patterns, enables more effective propagation of long-range dependencies and substantially boosts performance on long-sequence tasks. Meanwhile, we further refine NSA's branches with Latent Attention that the sliding-window branch is enhanced with Multi-head Latent Attention (MLA) while compression and selective branches adopt Group-head Latent Attention (GLA). These changes reduce KV-cache memory by 50% versus NSA while improving the model's common-sense reasoning and long-text understanding capabilities. Experiments on models from 340M to 1.3B parameters (trained on 15B and 100B tokens) show our method matches or exceeds full attention and native sparse attention in both common-sense reasoning and long-context understanding tasks.

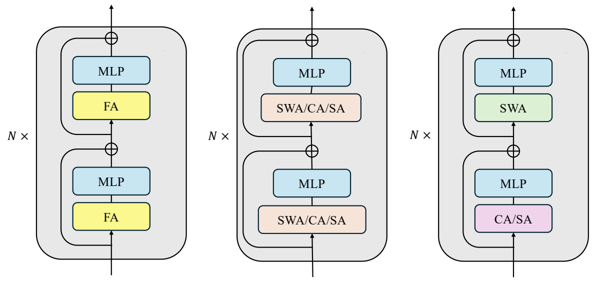

Figure 1: Overview of Llama-like Models Incorporating Full Attention (left), Native Sparse Attention (middle), and Alternating Sparse Attention (right). Here, FA denotes Full Attention, SWA denotes Sliding Window Attention, CA denotes Compressed Attention, and SA denotes Selective Attention.

1 Introduction

Benefiting from the application of transformer-based large language models (DeepSeek-AI et al., 2025b; Team et al., 2025a) (LLMs) in deep reasoning (DeepSeek-AI et al., 2025a; Team et al., 2025b), multi-turn agent systems, and codebase-level code comprehension, the research community has been increasingly focusing on the capabilities of LLMs in handling long-context inputs and test-time computation. As sequence lengths increase, the high computational complexity of the attention module has emerged as a significant efficiency bottleneck, constraining the further development of LLMs and necessitating the design of more efficient model architectures.

Given the inherent sparsity of softmax attention (Song et al., 2024), sparse attention has emerged as a promising strategy to accelerate attention computations, especially under long-context scenarios. In such cases, only a subset of critical key-value pairs interacts with the query at each generation step. Existing research has proposed a variety of training-free and post-training methods for selecting these key-value pairs, including KV-cache eviction techniques (Xiao et al., 2023), as well as index-based (Tang et al., 2024; Gao et al., 2025b), sampling-based (Chen et al., 2024), clustering-based (Liu et al., 2025), and hash-based approaches (Desai et al., 2024) for KV-cache selection. Despite their significant potential, these training-free and post-training techniques fail to fully explore model sparsity. To this end, recent research has introduced natively trainable sparse attention mechanisms (Lu et al., 2025; Yuan et al., 2025), among which native sparse attention (Yuan et al., 2025) (NSA) stands out as the most widely recognized and promising approach.

In the NSA framework, dense attention is decomposed into three components: sliding window attention, compressed attention, and selective at-

* Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 14209–14220
July 2-7, 2026

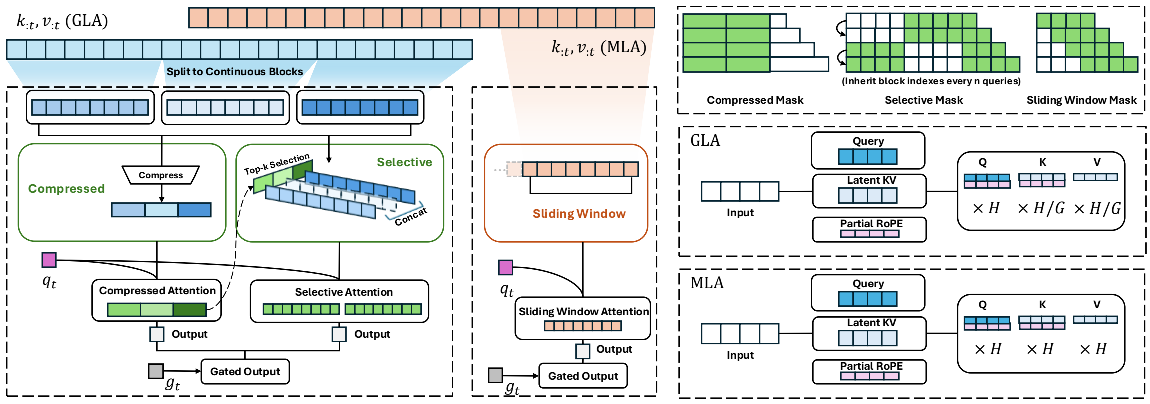

Figure 2: Overview of ASA's architecture. In ASA, consecutive attention layers alternate between compressed selective attention and sliding window attention. Furthermore, GLA and MLA replace the GQA mechanism used in NSA to enhance model expressiveness. To improve training efficiency, every consecutive 4 queries attend to the same key-value block.

tention. Through experimental analysis of these branches, we observe that sliding window attention plays the dominant role in common sense reasoning, while compression and selective attention primarily serve to enrich the model with global contextual information. Additionally, compared to using the same sparsity level across all layers, we found that an imbalanced sparsity distribution yields better performance on long retrieval tasks.

Building on this insight, we improve NSA by introducing targeted enhancements tailored to the distinct functional roles of each attention branch. The outcome is our proposed method, Alternating Sparse Attention (ASA), a sparse attention architecture explicitly designed for efficient and effective language modeling. As shown in Figure 1, ASA structures attention layers at the individual layer level into two complementary types: sliding window attention, which effectively models local contextual information, and compressed/selective attention, which efficiently captures long-range global context. These two type of layers are alternated in a 1:1, layer-wise pattern across layers, ensuring a balanced and synergistic representation of both local and global information throughout the model.

On the other hand, NSA is originally implemented based on grouped query attention (GQA). While GQA demonstrates strong effectiveness, certain attention mechanisms such as multi-head latent attention (MLA) comparable performance and substantially reducing GPU memory consumption. To further enhance NSA, we replaced GQA with MLA in ASA. However, MLA is equivalent to multi head attention during training, which presents challenges when integrating MLA with sparse attention mechanisms. To address this issue, we introduce a grouping mechanism into MLA, enabling it to better adapt to sparse attention mechanisms.

We conduct a thorough evaluation of ASA using transformer models with 340M and 1.3B parameters, trained on 15B and 100B tokens sampled from the SlimPajama dataset. The trained models are evaluated across multiple task categories, including general common sense reasoning, long-context retrieval and long-context understanding. Experimental results show that ASA achieves performance on par with, or even exceeds, that of the full attention baseline, while outperforming existing sparse attention approaches. Moreover, compared with the full-attention baseline, ASA reduces KV-cache storage overhead by 50% by applying sliding window attention (SWA) to only half of the Transformer layers, delivering substantial memory efficiency gains while maintaining model quality.

2 Related Work

With the increasing demand for processing long sequences in large language models, the quadratic complexity of vanilla attention has become a significant bottleneck. To address this challenge, a wide range of efficient attention mechanisms have been proposed (Sun et al., 2025), where sparse attention is considered a promising approach.

Sparse attention mechanisms aim to enhance computational efficiency by selectively computing attention scores over a strategically chosen subset of key-value pairs, rather than exhaustively attending to all possible token interactions. By exploiting the intrinsic sparsity observed in attention distributions, these approaches substantially reduce computational complexity, often achieving sub-quadratic scaling, while preserving the model's capacity to capture long-range contextual dependencies. Common sparse attention paradigms include attention sinks (Xiao et al., 2023), sliding window attention (Fu et al., 2025), selective attention, and hybrid architectures that combine multiple sparsity patterns.

A salient feature of many sparsity-based methods is their training-free design, which allows seamless integration with pre-trained dense models without requiring additional fine-tuning. For instance, techniques such as Streaming LLM (Xiao et al., 2023), SnapKV (Li et al., 2024), and PyramidKV (Cai et al., 2025) enforce a fixed-size cache during autoregressive decoding; as new key-value states are generated, they dynamically evict less salient states based on learned or heuristic attention scores. In contrast, methods like Quest (Tang et al., 2024), infLLM (Xiao et al., 2024), ClusterKV (Liu et al., 2025), and RetroInfer (Chen et al., 2025) retain the full sequence of key-value states and construct auxiliary indices to enable efficient retrieval. At each decoding step, only the most contextually relevant states are retrieved and attended to, thereby maintaining high fidelity while reducing computational overhead.

Further advances include approaches such as SeerAttention (Gao et al., 2025b) and SeerAttention-R (Gao et al., 2025a), which introduce sparsity during the fine-tuning phase by incorporating distillation objectives that align dense softmax attention with sparse approximations. This regularization encourages the model to learn to prioritize semantically critical key-value pairs, thereby improving both efficiency and retrieval accuracy. Most recently, natively trainable sparse attention architectures, such as MoBA (Lu et al., 2025) and NSA (Yuan et al., 2025), have been proposed to explicitly optimize sparsity patterns during training, offering a more principled and adaptive exploration of attention sparsity beyond static or heuristic designs.

Among them, although NSA achieves performance comparable to Full Attention while improving efficiency, it adopts a more complex attention architecture, a hybrid of window attention, compressed attention, and selective attention. Therefore, this work further analyzes the functional roles of the different attention branches and, based on these insights, proposes the improvement of NSA.

3 Background

Attention Mechanism. The attention mechanism constitutes a fundamental component of contemporary language models. Given a query q_t, it computes relevance scores with respect to all preceding keys $K_t = [k_1, k_2, \dots, k_t]$, which are subsequently utilized to generate a weighted sum over the corresponding values $V_t = [v_1, v_2, \dots, v_t]$. Formally, for an input sequence comprising t tokens, the attention mechanism with H query heads and group size G can be expressed as follows:

$$\begin{aligned} o_t &= \mathrm{Attn}(q_t, K_t, V_t, W_o) \\ &= \sum_{h=1}^{H} \mathrm{Softmax}\left(\frac{q_t^h K_t^{\lfloor \frac{h}{G} \rfloor \top}}{\sqrt{d_k}}\right) V_t^{\lfloor \frac{h}{G} \rfloor} W_o^h \end{aligned}$$

where $q_t \in R^{H \times d_k}, K_t \in R^{t \times \frac{H}{G} \times d_k}, V_t \in R^{t \times \frac{H}{G} \times d_v}, W_o^h \in R^{d_v \times d}$, d_k, d_v, and d represent the features dimension of keys, values, and hidden states respectively. It is evident that, within the attention mechanism, each query necessitates computation with all preceding key-value pairs. As the sequence length increases, the computational cost of attention progressively becomes the dominant factor in the overall model complexity, thereby presenting significant challenges for the efficient processing of long sequences.

Native Sparse Attention. To alleviate computational and memory access overhead in long-text scenarios, NSA introduces a native sparse attention mechanism. This mechanism decomposes the conventional attention operation into three distinct branches: sliding window attention (SWA), compressed attention (CA)), and selective attention (SA). Formally, NSA is defined as follows:

$$\begin{aligned} o_t &= g_t^{\text{swa}} o_t^{\text{swa}} + g_t^{\text{ca}} o_t^{\text{ca}} + g_t^{\text{sa}} o_t^{a} \\ o_t^{\text{swa}} &= \mathrm{Attn}(q_t, K_{t-s:t}, V_{t-s:t}, W_o) \\ o_t^{\text{ca}} &= \mathrm{Attn}(q_t, \hat{K}_t, \hat{V}_t, W_o) \\ o_t^{\text{sa}} &= \mathrm{Attn}(q_t, K_{I_t}, V_{I_t}, W_o) \end{aligned}$$

where g^{swa}, g^{ca} and g^{sa} are three gate scores to combine three attention branches. Specifically, for sliding window attention, $K_{t-s:t} = [k_{t-s}, \dots, k_t]$, where s denotes the sliding window size. For compressed attention, $\hat{K}_t = [\mathrm{ca}(k_{1:B}), \dots, \mathrm{ca}(k_{mB-B+1:mB})]$, where ca represents the compression operation, B is the compression block size, and $m = \lfloor \frac{t}{B} \rfloor - 1$. For selective attention, $K_{I_t} = \{K_{iB-B+1:iB}\}_{i \in I_t}$, where $I_t = \text{Top-K}(\mathrm{score}(q_t, \hat{K}_t))$ identifies the top K blocks based on the relevance scores between q_t and the compressed keys $\hat{K}_t$. $V_{t-s:t}$, $\hat{V}_t$, and V_{I_t} can be obtained using the same method as $K_{t-s:t}$, $\hat{K}_t$, and K_{I_t}.

Multi-head Latent Attention. The Multi-head Latent Attention (MLA) mechanism was first introduced in DeepSeek-V2 (DeepSeek-AI et al., 2024) and has demonstrated superior performance compared to conventional Multi-head Attention (MHA). MLA's key innovation lies in its use of latent states and a reparameterization strategy that allows it to emulate MHA during training while effectively operating as Multi-query Attention (MQA) during inference. Formally, given an input x, MLA first compresses x into a set of low-dimensional latent states c, where $\dim(c) \ll \dim(x)$. These latent states are then linearly projected to obtain key-value pairs: $K^h = W_k^h c$ and $V^h = W_v^h c$. Letting $q^h = W_q^h x$, MLA can be expressed as follows:[1]:

$$
\begin{aligned}
o_t &= \sum_{h=1}^{H} \left(\mathrm{Softmax}\left(q_t^h (c_{\le t} W_k^h)^\top \right) (c_{\le t} W_v^h) \right) W_o^h \\
&= \sum_{h=1}^{H} \mathrm{Softmax}\left((q_t^h W_k^{h\top}) c_{\le t}^\top \right) c_{\le t} (W_v^h W_o^h).
\end{aligned}
$$

By merging W_q^h and W_k^h into a single projection, and similarly combining W_v and W_o, MLA requires storing only the compact latent states c during decoding, effectively reducing its memory footprint to that of MQA while retaining the expressive power of MHA during training.

4 Rethinking Native Sparse Attention

In this section, we provide a detailed analysis of the individual functions of these branches as well as their combinatorial effects within the NSA framework.

4.1 Attention Modules Functional Analysis

Empirical evaluations of NSA demonstrate that its sparse attention formulation consistently achieves lower language modeling losses compared to full attention baselines. This observation naturally invites deeper inquiry: What are the distinct functional roles of each of the three sparse attention branches within the NSA architecture? Moreover, which branch contributes most substantially to overall performance?

To systematically evaluate the functional contributions of individual sparse attention branches within the NSA framework, we train three 340M-parameter models on a 15B-token corpus: (1) the full NSA architecture, (2) NSA with the sliding window branch ablated, and (3) NSA with the selective attention branch ablated. In addition, we examine the impact of directly removing attention branches from a pre-trained NSA model on downstream performance: (4) removal of the sliding window branch from the pre-trained NSA, and (5) removal of the selective attention branch from the pre-trained NSA. Finally, to assess the role of compressed attention, we first remove the selective attention branch and then train the NSA model under varying block sizes: (6) NSA without selective attention using a block size of 8, and (7) NSA without selective attention using a block size of 16. The experimental results are presented in Table 1, and further details of the experimental setup are provided in Section 6.

Based on these experiments, we draw the following conclusions: (a) Sliding window attention primarily impacts the model's performance on common-sense reasoning tasks. (b) Selective attention plays a crucial role in enhancing retrieval capabilities. (c) Compressed attention in NSA primarily functions as a supplementary mechanism to selective attention. (d) The concurrent use of sliding window and selective attention appears to diminish the retrieval capability of the selective attention branch.

By comparing experiments (1) and (4), we observe that removing window attention causes a significant decline in the model's performance metrics on common-sense reasoning tasks, corroborating conclusion (a). [2] Comparing experiments (1), (3), and (5) reveals that removing selective attention, either during pretraining or after training, leads to a significant drop in in-context retrieval task

[1] For the sake of conciseness, we omit the MLA's handling of positional encoding.

[2] Although experiment (2) indicates that removing window attention during pretraining has negligible impact on the model, further analysis reveals that some selective attention mechanisms degrade into window attention at pretraining stage.

Model	LAMB. ppl ↓	LAMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	BoolQ acc ↑	Avg.
NSA	44.34	31.03	64.31	34.78	51.07	44.07	23.29	58.06	43.80
Train from scratch									
- *(w/o. sa)*	40.58	31.24	63.66	34.70	52.33	43.98	22.70	55.81	43.49
- *(w/o. swa)*	39.42	32.14	63.98	35.07	50.83	44.28	22.70	57.25	43.75
- *(w. alt)*	40.47	31.07	64.25	34.58	52.25	43.14	22.87	58.44	43.80
Traning free									
- *(w/o. sa)*	227.0	12.58	63.00	33.51	51.85	41.58	23.98	62.08	41.23
- *(w/o. swa)*	NAN	0.0	51.85	25.90	49.96	25.97	26.79	38.26	31.25
NSA w/o. sa									
- *(block size = 8)*	40.23	31.71	63.93	35.27	52.80	43.01	23.46	57.58	43.97
- *(block size = 16)*	40.58	31.24	63.66	34.70	52.33	43.98	22.70	55.81	43.49

Model	S-NIAH-1 (pass-key retrieval)			S-NIAH-2 (number in haystack)			S-NIAH-3 (uuid in haystack)		
	2k	4k	8k	2k	4k	8k	2k	4k	8k
NSA	100.0	100.0	99.0	100.0	98.0	52.2	79.2	43.2	11.6
Train from scratch									
- *(w/o. swa)*	100.0	100.0	95.60	100.0	92.6	53.4	99.4	58.0	30.6
- *(w/o. sa)*	27.6	12.6	6.4	30.2	17.2	8.0	31.6	14.4	7.4
- *(w. alt)*	100.0	100.0	100.0	100.5	100.0	97.8	83.4	55.2	22.0
Traning free									
- *(w/o. swa)*	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
- *(w/o. sa)*	23.6	10.2	4.8	20.0	9.0	3.8	3.2	0.8	0.8
NSA w/o. sa									
- *(block size = 8)*	28.2	13.0	6.8	31.8	18.6	8.0	30.8	14.6	8.0
- *(block size = 16)*	27.6	12.6	6.4	30.2	17.2	8.0	31.6	14.4	7.4

Table 1: Ablation results of NSA on common-sense reasoning and in-context retrieval benchmarks. *NSA (w/o. swa)*, *NSA (w/o. sa)*, and *NSA (w. alt)* denote removing sliding window attention, removing selective attention, and modifying NSA to alternately use sliding window attention and selective attention, respectively.

metrics, confirming conclusion (b). Comparing experiments (1), (6), and (7) shows that even with finer-grained compressed attention, removing selective attention fails to effectively improve in-context retrieval performance, supporting conclusion (c). Finally, comparing experiments (1) and (2) reveals that removing window attention actually enhances the model's in-context retrieval capability, confirming conclusion (d). We hypothesize that this occurs because the sliding window attention mechanism is more readily learned by the model, forming a shortcut that reduces reliance on selective attention during retrieval tasks, thereby weakening its effectiveness.

4.2 Attention Modules Combination Analysis

Excluding combinations of the three attention branches, the NSA framework applies a uniform sparsity rate to the selective attention branch at each layer, resulting in the retrieval of an identical number of key-value blocks. This design choice raises an important research question: Is an even distribution of sparsity across all layers optimal for selective attention?

Motivated by recent developments in hybrid attention architectures in open-source models (Puvvada et al., 2025), we investigated two alternative combination strategies: (1) Alternating between compressed/selective attention and sliding window attention, i.e., removing selective attention from half of the layers and consolidating their computational workload into the remaining layers. (2) Applying the same sparsity level to the selective attention branches across all layers.

The experimental results are also shown in Table 1. It is evident that the alternating sliding window attention and selective attention (NSA with alt.) achieves superior contextual retrieval performance relative to the standard NSA architecture, while preserving comparable common-sense reasoning abilities. This suggests that employing a non-uniform sparsity strategy across different layers yields better performance than applying a uniform sparsity level consistently across all layers. Additionally, this approach reduces the storage overhead of the KV-Cache by half.

5 Methodology

In this section, we introduce our proposed method, Alternating Sparse Attention (ASA) and detail the algorithmic and engineering advancements incorporated in ASA over NSA, leading to improved model performance as well as enhanced training and inference efficiency.

5.1 Alternating Sparse Attention

In the preceding sections, our analysis of the various branches within NSA leads to the following conjectures: (1) sliding window attention plays a predominant role in minimizing language modeling loss. (2) The selective branch primarily facilitates long-context retrieval. (3) While the selective branch is essential, it is not required for all layers; employing computationally intensive selective branches for a subset of layers yields better performance than applying lightweight selective branches uniformly across all layers.

Based on the conjecture above, we propose the Alternating Sparse Attention (ASA) mechanism, which introduces several architectural refinements to the baseline NSA framework, enhancing efficiency, scalability, and modeling capacity. Our improvements are structured as follows:

First, we redistribute the three attention branches originally integrated within each NSA layer across distinct layers of the model. This stratification ensures that each attention head specializes in a single sparsity pattern, thereby reducing interference and improving representational focus. Specifically, within each transformer layer, we sequentially apply the selective/compressed attention branch followed by the sliding window attention branch.

Second, considering the superior efficiency of MLA over GQA, we replace GQA with MLA in the ASA module. For the sliding window attention branch of ASA, the MHA-based training regime of MLA endows it with superior representational capacity compared to GQA. Meanwhile, its inference-stage adoption of MQA ensures high computational efficiency without compromising performance.

Nevertheless, while the MHA training paradigm is well-suited for the sliding window attention branch, it fundamentally conflicts with the compressed and selective attention: in MHA, each query head employs independent key and value projection matrices. In contrast, the NSA compute kernel imposes a hardware-level constraint that every contiguous block of at least 16 query heads must share identical key and value projections. To reconcile this discrepancy, we introduce a structured grouping mechanism within MLA, specifically in the compressed and selective attention branch, wherein query heads are explicitly grouped to share common key and value projections. This design preserves training flexibility while satisfying the NSA kernel's alignment requirement. This adaptation gives rise to Grouped-head Latent Attention (GLA). Formally, GLA can be expressed as follow:

$$\mathrm{GLA}(q_t, c_{\leq t}) = \sum_{i=1}^{H/G} \sum_{j=1}^{G}$$
$$\mathrm{Softmax}\big(q_t^{iG+j}(c_{\leq t} W_k^j)^\top\big) c_{\leq t} W_v^j W_o^{iG+j},$$

where H denotes the number of attention heads, q_t represents the query at time step t, and $c_{\leq t}$ denotes the latent states up to time t, G is the group size, and each group of G heads shares the same key and value projection matrices (W_k^j and W_v^j), while retaining distinct output projections (W_o^{iG+j}). In Appendix A, we provide PyTorch-style pseudocode for ASA.

5.2 Kernel Optimization

The progress of sparse attention has been limited by the lack of hardware-efficient kernels, which makes it challenging to use sparse attention during the pre-training of models. To enable efficient pre-training with sparse attention, NSA modifies flash attention and introduces a kernel that is optimized for hardware efficiency in sparse attention scenarios. The main change in NSA is to move the partitioning strategy from the query sequence dimension to the head number dimension. Specifically, in the NSA kernel, each compute unit first loads the query matrix of shape $[G, d_k]$, and then sequentially loads the key and value matrices, each of shape $[B, d_k]$ and $[B, d_v]$ to perform attention computation. Consequently, the parallelism of the NSA kernel is constrained by the group size G.

Prior studies have shown that, in sparse attention mechanisms, the sets of key-value blocks re-

Model	LAMB. ppl ↓	LAMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	BoolQ acc ↑	Avg.
340M params									
GQA	36.40	34.33	62.95	34.65	50.99	43.73	23.55	52.51	43.24
NSA	44.34	31.03	64.31	34.78	51.07	44.07	23.29	58.06	43.80
ASA	40.47	31.07	64.25	34.86	52.25	44.28	23.29	58.44	44.06
1.3B params									
GQA	14.99	47.56	69.31	49.59	54.54	55.30	26.96	56.91	51.45
NSA	12.29	50.44	71.06	51.67	55.56	57.07	26.71	58.20	52.96
ASA	11.21	51.73	71.33	51.73	55.01	56.52	27.13	58.26	53.10

Table 2: Experiments results of GQA, NSA and ASA on common-sense reasoning benchmarks.

Model	S-NIAH-1 (pass-key retrieval)			S-NIAH-2 (number in haystack)			S-NIAH-3 (uuid in haystack)		
	2k	4k	8k	2k	4k	8k	2k	4k	8k
340M params									
GQA	100.0	100.0	100.0	100.0	83.2	54.6	97.2	90.8	33.0
NSA	100.0	100.0	99.0	100.0	98.0	52.2	79.2	43.2	11.6
ASA	100.0	100.0	100.0	100.0	100.0	99.8	83.4	62.6	52.6
1.3B params									
GQA	100.0	100.0	100.0	100.0	100.0	100.0	84.2	93.0	64.4
NSA	100.0	99.8	98.8	100.0	99.8	66.0	89.6	78.8	65.0
ASA	100.0	100.0	100.0	100.0	100.0	100.0	87.4	79.4	62.0

Table 3: Experiments results of GQA, NSA and ASA on in-context retrieval benchmarks.

trieved by neighboring queries often exhibit substantial overlap. Inspired by this observation, we propose an optimization: grouping every consecutive 4 queries to attend to the same key-value block. This allows each compute unit to load a query matrix of shape $[4G, d_k]$ and compute attention with the shared key-value block in a more parallelized manner, thereby improving kernel-level parallelism. Compared to NSA, this design enables ASA to utilize computational resources more efficiently. Empirical results further demonstrate that this modification incurs only negligible performance degradation. The impact of kernel optimization on both computational efficiency and model performance is presented in the Appendix B.

6 Experiments

Following the common practice in existing works, we evaluate ASA through common-sense reasoning tasks, in-context retrieval tasks, and long-context understanding tasks, comparing against group-query attention and native-sparse attention baseline.

Setup In our experiments, to ensure a fair comparison, all models were trained under identical conditions. We adopted the Llama architecture as the backbone and developed two model variants with 340M and 1.3B parameters, respectively. Detailed parameter configurations are shown in Table 5. The models were trained on 15B and 100B tokens, which were sampled from the SlimPajama dataset. For optimization, we utilized the AdamW optimizer with a peak learning rate of 3×10^{-4} and a minimum learning rate of 3×10^{-5}, a weight decay coefficient of 0.01, and a gradient clipping threshold of 1.0. The learning rate was maintained using a cosine schedule, with a warm-up period of 0.5B tokens, and a batch size of 0.5M tokens. All models utilized the Llama-2 (Touvron et al., 2023) tokenizer, which has a vocabulary size of 32,000. During training, the maximum context length was set to 8K tokens. To accelerate training, the key-value block retrieved from the first token is reused for every 4 consecutive tokens in ASA. For compression and selective attention, the block size is set to 16. Also, 64 blocks are selected per NSA layer, while 128 blocks are selected per pair of ASA layers. For comparison, we evaluate against the group query attention (Ainslie et al., 2023) and native sparse attention (Yuan et al., 2025).

Common-Sense Reasoning To assess the model's

Model	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Code		Avg
	NQA	QQA	MFQ	HQA	2WM	Mus	GvR	QMS	MNs	TRC	TQA	SSM	LCC	RBP	
340M params															
GQA	2.68	5.87	9.80	3.62	6.14	2.03	19.71	14.33	17.33	21.00	12.16	10.15	13.22	12.39	10.75
NSA	2.69	5.75	9.59	2.46	6.07	1.66	18.90	13.94	17.87	19.50	10.10	4.99	19.95	20.87	11.02
ASA	2.82	6.25	10.64	3.95	6.20	2.45	16.47	14.23	18.01	33.00	12.96	8.43	21.72	20.19	12.67
1.3B params															
GQA	2.92	7.77	13.52	5.53	8.80	3.13	21.65	15.22	20.33	37.50	32.34	18.68	22.72	20.81	16.49
NSA	3.83	6.94	12.74	5.40	7.56	2.29	22.48	15.21	16.80	38.50	29.21	16.38	28.38	29.29	16.78
ASA	3.39	8.35	14.29	4.46	8.09	2.96	23.51	15.89	18.71	54.00	26.87	16.47	27.49	30.96	18.25

Table 4: Experiment results of GQA, NSA and ASA on long-context understanding benchmarks.

	340M		1.3B	
	GQA/NSA	ASA	GQA/NSA	ASA
n_{layer}	21		24	
d_{model}	1024		2048	
h_{q}	16		32	
h_{kv}	1		2	
d_{vo}	128		128	
d_{ffn}	2816		5632	
h_{latent}	-	256	-	512
d_{qk}	128	192	128	192

Table 5: Parameter configuration for GQA, NSA, and ASA models with 340M/1.3B parameters.

common-sense reasoning capabilities, we follow previous works (Yang et al., 2024) and evaluate our model as well as baselines on several widely-used benchmarks. These include PIQA (Bisk et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2019), ARC-easy and ARC-challenge (Clark et al., 2018), SIQA (Sap et al., 2019), BoolQ (Clark et al., 2019), and LAMBADA (Paperno et al., 2016).

In-Context Retrieval For in-context retrieval tasks, we employ the Needle-In-A-Haystack Single (NIAH-S) benchmark from RULER (Hsieh et al., 2024), which comprises three tasks of increasing complexity: S-NIAH-1 (passkey retrieval), S-NIAH-2 (numerical needle in haystack), and S-NIAH-3 (word-based needle in haystack).

Long Context Understanding To evaluate long context understanding, we use 14 tasks from the LongBench (Bai et al., 2024). These tasks cover various aspects, including narrative comprehension (Kočiský et al., 2017) (Narrative QA), scientific understanding (Dasigi et al., 2021) (QasperQA), multi-hop reasoning (MultiField QA, Hotpot QA (Yang et al., 2018), 2WikiMulti QA (Ho et al., 2020), Musique (Trivedi et al., 2022)), document summarization (GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), Multi-News (Fabbri et al., 2019)), as well as specialized tasks such as TRec (Li and Roth, 2002), Trivia QA (Joshi et al., 2017), SAMSum (Gliwa et al., 2019), LCC (Mohler et al., 2016), and RepoBench-P (Liu et al., 2023).

Experiment Results The experimental results of ASA and the baseline methods on common-sense reasoning, in-context retrieval, and long-context understanding benchmarks are presented in Tables 2, 3, and 4.

For common-sense reasoning tasks, ASA achieves slightly better performance than both GQA and NSA, highlighting the gains brought by replacing GQA with MLA. In context retrieval tasks, ASA clearly outperforms NSA, benefiting from the use of alternating hybrid window attention and selective attention. Additionally, with the integration of GLA, which increases the key-value dimensions during attention computation, ASA is able to surpass even the GQA baseline on the S-NIAH-2 task. On the S-NIAH-3 task, although ASA falls short of GQA at the 4k context length, it outperforms GQA at the 8k length. Finally, for long-context understanding tasks, ASA consistently outperforms both GQA and NSA across nearly all benchmarks.

7 Conclusion

In this work, we propose Alternating Sparse Attention (ASA), a novel sparse attention architecture in which sliding window attention and compressed/selective attention are alternated across layers, and further enhanced with multi-head and group-head latent attention mechanisms, respectively. The kernel is optimized along both the query head dimension and the query sequence dimension, thereby improving kernel-level parallelism. ASA achieves efficient long-context modeling for transformer-based large language models, matching or surpassing GQA and NSA in performance while reducing

KV-cache storage by 50%, thus providing a practical and scalable solution for language modeling.

8 Limitation

Although this work demonstrates that applying hybrid window attention and compressed selective attention in different layers yields better model performance than using them concurrently within the same layer, further exploration of optimal blending strategies remains necessary. For instance, it would be valuable to investigate whether there exists an optimal ratio and placement scheme for integrating hybrid window attention with compression and selective attention. Moreover, it also merits further study whether substituting sliding window attention with modern linear attention architectures could lead to models with greater expressive power.

Acknowledgments

This work is supported by the National Key Research & Development Plan (2023YFF0725100) and the National Natural Science Foundation of China (92570121, 62322214, U23A20299, U24B20144).

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *Preprint*, arXiv:2305.13245.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. *Preprint*, arXiv:2308.14508.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language. *Preprint*, arXiv:1911.11641.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2025. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *Preprint*, arXiv:2406.02069.

Yaoqi Chen, Jinkai Zhang, Baotong Lu, Qianxi Zhang, Chengruidong Zhang, Jingjia Luo, Di Liu, Huiqiang Jiang, Qi Chen, Jing Liu, Bailu Ding, Xiao Yan, Jiawei Jiang, Chen Chen, Mingxing Zhang, Yuqing Yang, Fan Yang, and Mao Yang. 2025. Retroinfer: A vector-storage approach for scalable long-context llm inference. *Preprint*, arXiv:2505.02922.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and 1 others. 2024. Magicpig: Lsh sampling for efficient llm generation. In *The Thirteenth International Conference on Learning Representations*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *Preprint*, arXiv:1905.10044.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *Preprint*, arXiv:2105.03011.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, and 1 others. 2025a. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, and 1 others. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, and 1 others. 2025b. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Aditya Desai, Shuo Yang, Alejandro Cuadron, Matei Zaharia, Joseph E Gonzalez, and Ion Stoica. 2024. Hashattention: Semantic sparsity for faster inference. In *Forty-second International Conference on Machine Learning*.

Alexander R. Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R. Radev. 2019. Multi-news: a large-scale multi-document summarization dataset and abstractive hierarchical model. *Preprint*, arXiv:1906.01749.

Zichuan Fu, Wentao Song, Yejing Wang, Xian Wu, Yefeng Zheng, Yingying Zhang, Derong Xu, Xuetao Wei, Tong Xu, and Xiangyu Zhao. 2025. Sliding window attention training for efficient large language models. *Preprint*, arXiv:2502.18845.

Yizhao Gao, Shuming Guo, Shijie Cao, Yuqing Xia, Yu Cheng, Lei Wang, Lingxiao Ma, Yutao Sun, Tianzhu Ye, Li Dong, Hayden Kwok-Hay So, Yu Hua, Ting Cao, Fan Yang, and Mao Yang. 2025a.

Seerattention-r: Sparse attention adaptation for long reasoning. *Preprint*, arXiv:2506.08889.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, and Mao Yang. 2025b. Seerattention: Learning intrinsic sparse attention in your llms. *Preprint*, arXiv:2410.13276.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China. Association for Computational Linguistics.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *Preprint*, arXiv:2011.01060.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *Preprint*, arXiv:2404.06654.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. *Preprint*, arXiv:2104.02112.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *Preprint*, arXiv:1705.03551.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2017. The narrativeqa reading comprehension challenge. *Preprint*, arXiv:1712.07040.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *International Conference on Computational Linguistics*.

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. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. 2025. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pages 1–7. IEEE.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023. Repobench: Benchmarking repository-level code auto-completion systems. *Preprint*, arXiv:2306.03091.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, and 6 others. 2025. Moba: Mixture of block attention for long-context llms. *Preprint*, arXiv:2502.13189.

Michael Mohler, Mary Brunson, Bryan Rink, and Marc Tomlinson. 2016. Introducing the LCC metaphor datasets. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4221–4227, Portorož, Slovenia. European Language Resources Association (ELRA).

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. *Preprint*, arXiv:1606.06031.

Krishna C. Puvvada, Faisal Ladhak, Santiago Akle Serrano, Cheng-Ping Hsieh, Shantanu Acharya, Somshubra Majumdar, Fei Jia, Samuel Kriman, Simeng Sun, Dima Rekesh, and Boris Ginsburg. 2025. Swan-gpt: An efficient and scalable approach for long-context language modeling. *Preprint*, arXiv:2504.08719.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *Preprint*, arXiv:1907.10641.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *Preprint*, arXiv:1904.09728.

Zhao Song, Jing Xiong, and Chiwun Yang. 2024. How sparse attention approximates exact attention? your attention is naturally nc-sparse. In *Sparsity in LLMs (SLLM): Deep Dive into Mixture of Experts, Quantization, Hardware, and Inference*.

Yutao Sun, Zhenyu Li, Yike Zhang, Tengyu Pan, Bowen Dong, Yuyi Guo, and Jianyong Wang. 2025. Efficient attention mechanisms for large language models: A survey. *Preprint*, arXiv:2507.19595.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. Quest: Query-aware sparsity for efficient long-context llm inference. In *International Conference on Machine Learning*, pages 47901–47911. PMLR.

Meituan LongCat Team, Bayan, Bei Li, Bingye Lei, Bo Wang, Bolin Rong, Chao Wang, Chao Zhang, Chen Gao, Chen Zhang, Cheng Sun, and 1 others. 2025a. Longcat-flash technical report. *Preprint*, arXiv:2509.01322.

Meituan LongCat Team, Anchun Gui, Bei Li, Bingyang Tao, Bole Zhou, Borun Chen, Chao Zhang, Chao Zhang, Chengcheng Han, Chenhui Yang, Chi

Zhang, Chong Peng, Chuyu Zhang, Cong Chen, Fengcun Li, Gang Xu, and 1 others. 2025b. Longcat-flash-thinking technical report. *Preprint*, arXiv:2509.18883.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Preprint*, arXiv:2108.00573.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. 2024. Infllm: Training-free long-context extrapolation for llms with an efficient context memory. *Advances in Neural Information Processing Systems*, 37:119638–119661.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. 2024. Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Preprint*, arXiv:1809.09600.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. Qmsum: A new benchmark for query-based multi-domain meeting summarization. *Preprint*, arXiv:2104.05938.

A PyTorch-style pseudocode for ASA

In Lsting 1, we present the pytorch-style pseudocode for ASA. In this code, the input to the attention module is denoted by x. The matrices $W_{c/q/k/v/g}$ represent learnable weights. Here, T refers to the number of tokens, HQ and H represents the number of attention heads for queries and key-values, d denotes the hidden dimension of the input, d_c represents the hidden dimension of the latents, d_p indicates the hidden dimension of the partial RoPE, and d_k and d_v correspond to the hidden dimensions of the key and value, respectively.

Listing 1: PyTorch-style pseudocode for ASA

```
def ASA_sliding_window_attention(
    x, W_q, W_c, W_p, W_k, W_v, W_g,
    T, HQ, d_c, d_p, d_k, d_v,
):
    q = (x @ W_q).view(T, HQ, d_k)
    q_nope, q_rope = split(q, [d_k - d_p, d_p], dim=-1)
    q_rope = RoPE(q_rope)
    q = cat([q_nope, q_rope], dim=-1)
    k_rope = RoPE((x @ W_p).view(T, 1, d_p))
    c = (x @ W_c).view(T, 1, d_c)
    k_nope = (c @ W_k).view(T, HQ, d_k - d_p)
    k = cat([k_nope, repeat(k_rope)], dim=-1)
    v = (c @ W_v).view(T, HQ, d_v)
    g = (c @ W_g).view(T, HQ)
    o = g * sliding_window_attention(q, k, v)
    return o

def ASA_caressed_selected_attention(
    x, W_q, W_c, W_p, W_k, W_v, W_g,
    T, B, HQ, H, d_c, d_p, d_k, d_v,
):
    q = (x @ W_q).view(T, HQ, d_k)
    q_nope, q_rope = split(q, [d_k - d_p, d_p], dim=-1)
    q_rope = RoPE(q_rope)
    q = cat([q_nope, q_rope], dim=-1)
    k_rope = RoPE((x @ W_p).view(T, 1, d_p))
    c = (x @ W_c).view(T, 1, d_c)
    k_nope = (c @ W_k).view(T, H, d_k - d_p)
    k = cat([k_nope, repeat(k_rope)], dim=-1)
    v = (c @ W_v).view(T, H, d_v)
    g_ca, g_sa = split((c @ W_g).view(T, HQ * 2))
    # [T/B, H, d_k/d_v]
    k_ca, v_ca = compress(k, v, B)
    I = topk(q, k_ca)
    k_sa, v_sa = select(k, v, I)
    o = g_ca * compressed_attention(q, k_ca, v_ca) \
        + g_sa * selected_attention(q, k_sa, v_sa)
    return o
```

Model	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Code		Avg
	NQA	QQA	MFQ	HQA	2WM	Mus	GvR	QMS	MNs	TRC	TQA	SSM	LCC	RBP	
340M params															
NSA	2.69	5.75	9.59	2.46	6.07	1.66	18.90	13.94	17.87	19.50	10.10	4.99	19.95	20.87	11.02
NSA (w. KO)	2.54	6.05	9.59	2.93	5.00	2.13	19.72	14.39	20.13	12.75	15.30	8.50	17.13	16.99	10.94

Table 6: Experiment results of NSA and NSA with kernel optimization (w. KO) on long-context understanding benchmarks.

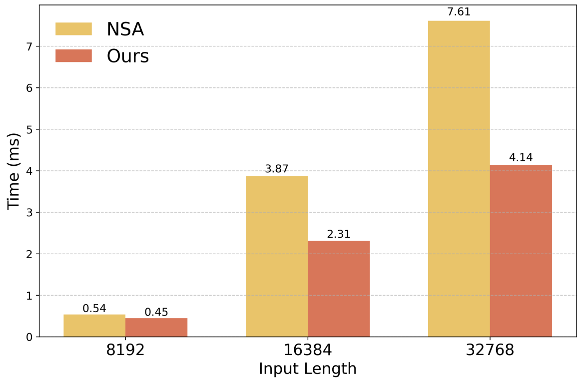

Figure 3: Computation time comparison for forward.

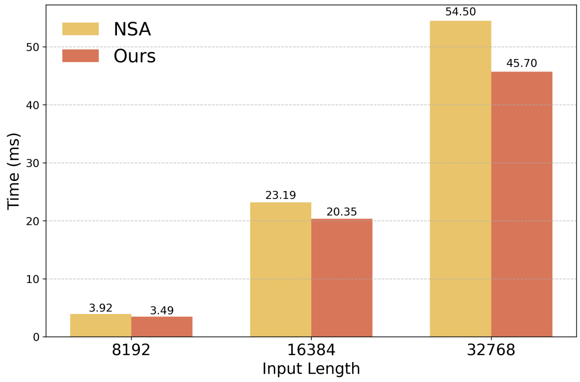

Figure 4: Computation time comparison for backward.

B Kernel Optimization

We implemented our improved kernel based on the open-source NSA kernel[3] and compared it with the original version, in which consecutive four queries are guaranteed to select the same block. The forward and backward computation times were measured for sequence lengths of 8192, 16384, and 32768. All evaluations were conducted on a single H800 GPU. In the experiments, the batch size was set to 1, the number of KV groups was 1, and each KV group contained 16 query heads. The block size was set to 16, and each query selected 64 blocks (i.e., 1024 tokens). The experimental results are shown in Figures 3 and 4. As can be seen, the optimized kernel reduces the forward computation time by approximately 30% and the backward computation time by about 13%. As shown in Table 6, we also evaluate NSA's performance on the long-context understanding dataset under two settings: with and without kernel optimization. The results suggest that applying kernel optimization results in only a negligible reduction in model performance.

[3] https://github.com/fla-org/native-sparse-attention