DSSA: DENSE-SPARSE SWITCHABLE ATTENTION FOR SEAMLESS SHORT-TO-LONG ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-sequence processing is a critical capability for modern large language models. However, the self-attention mechanism in the standard Transformer architecture faces severe computational and memory bottlenecks when processing long sequences. While trainable sparse attention methods offer a promising solution, existing approaches such as NSA introduce excessive extra parameters and disrupt the conventional *pretrain-on-short, finetune-on-long* workflow, resulting in slow convergence and difficulty in acceleration. To overcome these limitations, we introduce **D**ense-**S**parse **S**witchable **A**ttention framework (**DSSA**), a trainable sparse attention that seamlessly adapts models from short to long sequences. Specifically, DSSA reuses dense attention parameters through parameter-free architecture modification, maintaining consistency between short and long sequence processing. Additionally, DSSA ensures computational efficiency across all sequence lengths, by using dense attention for short inputs and smoothly transitioning to sparse attention for long sequences. To achieve practical acceleration, we further introduce an efficient implementation of DSSA that significantly reduces the computational overhead. Our experiments on long-context understanding and chain-of-thought reasoning demonstrate that DSSA is $4\times$ faster than dense attention while retaining 98.1% and 99.7% of the performance, respectively. We will release all associated implementations to facilitate future research on efficient attention.

1 INTRODUCTION

With the rapid development of large language models (LLMs) (Brown et al., 2020; Bommasani et al., 2021; Han et al., 2021; OpenAI, 2023), the demand for long-sequence processing capabilities has become increasingly critical. From long-input scenarios such as deep research (Zheng et al., 2025; Xu & Peng, 2025), chatbots with long-term memory, and software issue resolution (Jimenez et al., 2023; Yang et al., 2025), to long-output tasks including complex reasoning (OpenAI et al., 2024; DeepSeek et al., 2025) and LLM-driven agents (Wang et al., 2024), a model's capability to understand and generate long sequences directly determines its performance in real-world applications. However, the self-attention mechanism in the existing Transformer (Vaswani et al., 2017) architecture faces severe computational and memory bottlenecks when processing long sequences.

To address the challenge of processing long sequences, efforts have been devoted to exploring sparse attention mechanisms (Beltagy et al., 2020; Zaheer et al., 2020; Tay et al., 2022), which restrict each token within the context to attend to only a subset of tokens related to that token. Early research in this area focuses on the training-free setting, leveraging the sparsity naturally occurring in self-attention mechanisms to accelerate inference (Xiao et al., 2024a;b; Jiang et al., 2024). However, the training-free setting introduces a fundamental trade-off between sparsity and model performance. To avoid significant performance degradation, the degree of sparsity that can be applied is often limited, which in turn restricts the potential efficiency gains.

Given the limitations of training-free attention mechanisms, trainable sparse attention mechanisms have garnered increasing attention from researchers (Lu et al., 2025; Gao et al., 2024). Among them, the natively trainable sparse attention (NSA) (Yuan et al., 2025) method adopts the widely-used block-sparse attention (Child et al., 2019) structure, designing three different sparse attention modules and developing corresponding CUDA kernels to accelerate model computation. Despite its effectiveness, we find **misalignment between the sparse architecture of NSA and the stan-**

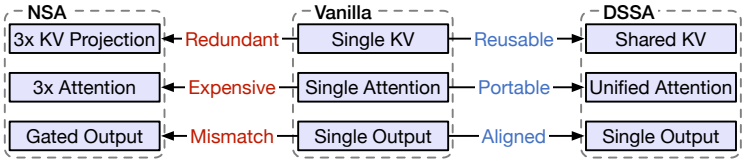

Figure 1: The comparison of Vanilla Full Attention, NSA (Yuan et al., 2025), and our DSSA.

dard pretrain-on-short, finetune-on-long workflow. A widely used way to build long LLMs is to pretrain on short sequences and finetune on long sequences. The NSA creates an architectural mismatch with vanilla full attention, as it introduces three sets of key-value parameters and three attention modules, forcing the model to abruptly switch from a single-output attention to a multi-output attention architecture. As shown in Section 4, this mismatch hinders smooth adaptation, erases what the model has already learned, and introduces a significant efficiency bottleneck for short sequences.

To address all the above issues, we propose **D**ense-**S**parse **S**witchable **A**ttention framework (**DSSA**). DSSA is built on block-sparse attention and introduces three core innovations:

1. **Seamless Short-to-Long Adaptation:** As depicted in Figure 1, different from NSA, which requires additional parameters and multiple attention modules, DSSA seamlessly transitions from dense to sparse attention by directly reusing existing dense attention parameters. This design naturally aligns with the standard pretrain-on-short, finetune-on-long workflow, eliminating architectural mismatches and minimizing loss fluctuations.
2. **Efficiency for Both Short and Long Sequences:** Because the transition from dense to sparse attention in DSSA requires no additional parameters and introduces minimal distributional shifts, the model preserves its strong performance on short texts and can easily switch back to dense attention for short sequence efficiency.
3. **Accelerated Block Selection Mechanism:** The block selection step before sparse attention inherently undermines the efficiency gains of the sparse attention itself. We propose a hardware-awared efficient implementation, effectively removing the bottleneck and unlocking the full potential of sparse attention.

We evaluate our method on long-context understanding and long chain-of-thought (CoT) generation benchmarks. Our DSSA is $4\times$ faster than dense attention while maintaining 98.1% and 99.7% of the original performance on these tasks, respectively. To advance research in sparse attention, we are releasing all operator implementations of DSSA.

2 Related Work

As the demand for LLMs to understand and generate long sequences continues to grow, research on improving attention efficiency has garnered increasing attention (Tay et al., 2022; Sun et al., 2025; Zhang et al., 2025a). In this section, we discuss the sparse attention paradigm from two perspectives: training-free and trainable sparse attention approaches.

2.1 Training-free Sparse Attention

Training-free sparse attention approaches aim to utilize the intrinsic sparsity of attention layers. These methods enable LLMs trained with dense attention to perform sparse attention between each token and a small subset of relevant contexts. Based on the selection strategy for relevant contexts, these algorithms can be categorized into predefined sparse patterns and dynamic sparse patterns.

Predefined Sparse Patterns. Sparse attention with a predefined pattern employs manually defined heuristic rules to determine which contextual tokens should be selected for attention computation (Xiao et al., 2024b; Han et al., 2024; Child et al., 2019; Zaheer et al., 2020; Beltagy et al., 2020; Xiao et al., 2025). For instance, sliding window attention restricts each token to interact only with neighboring tokens (Beltagy et al., 2020). Building upon sliding windows, some works select special tokens such as initial tokens or segment separators, requiring all tokens to attend to these special tokens (Xiao et al., 2024b; Chen et al., 2024; Child et al., 2019). Furthermore, some works combine

multiple predefined attention patterns (Zaheer et al., 2020; Beltagy et al., 2020).These approaches typically rely on human observations to formulate heuristic rules for selecting relevant contexts.

Dynamic Sparse Patterns. Dynamic sparse patterns incorporate the semantic information of query tokens into the context selection process by computing the relevance between query tokens and candidate contexts. Early works primarily perform similarity computation at the token level (Kitaev et al., 2020; Roy et al., 2021; Wang et al., 2020). As sequence lengths increase, block sparse methods have gained widespread adoption, which partition contexts into contiguous block units and perform relevance computation and context selection at the block granularity (Xiao et al., 2024a; Jiang et al., 2024; Xu et al., 2025; Tang et al., 2024; Zhang et al., 2025b; Lai et al., 2025). Furthermore, research on attention sparsity has inspired the development of key-value (KV) eviction and compression methods, which reduce memory consumption by discarding or compressing KV caches with low attention probabilities (Zhang et al., 2023; Li et al., 2024; Huang et al., 2024; 2025).

Training-free methods, while focusing on improving the inference efficiency of dense attention models, are often constrained by insufficient sparsity levels in order to avoid severe performance degradation and finally suffer from limited acceleration benefits.

2.2 Trainable Sparse Attention

To further enhance efficiency for long sequence processing, researchers incorporate sparse attention into the model training phase. SeerAttention (Gao et al., 2024) employs a self-distillation post-training algorithm to train a router that selects relevant contexts for query blocks. MoBA (Lu et al., 2025) employs a block sparse attention structure during the short-to-long adaptation phase, training routers between query blocks and KV blocks for context selection. These methods partition query tokens into blocks and can only accelerate the prefilling phase. NSA (Yuan et al., 2025) designs three attention components for token-level sparsity, effectively accelerating both prefilling and decoding processes. However, NSA introduces substantial additional parameters, making it unsuitable for efficient short-to-long adaptation and imposing significant computational overhead on short-sequence processing. In this paper, we focus on proposing a sparse attention mechanism that effectively and efficiently processes both short and long sequences, supporting both prefilling and decoding.

3 Method

3.1 Background

Grouped-Query Attention. Attention mechanisms enable models to selectively focus on relevant parts of the input sequence. Among various attention variants, grouped-query attention (GQA) (Ainslie et al., 2023) has emerged as a popular method that strikes a balance between model performance and computational efficiency. Given an input sequence of hidden states $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the sequence length and d is the model dimension, GQA computes the queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$) via linear projections as $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \mathbf{K} = \mathbf{X}\mathbf{W}_K, \mathbf{V} = \mathbf{X}\mathbf{W}_V$. The projection matrices have the shapes $\mathbf{W}_Q \in \mathbb{R}^{d \times (h_q d_h)}$ and $\mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times (h_{kv} d_h)}$, with the head dimension d_h. These tensors are then reshaped to form h_q query heads $\{\mathbf{Q}_i\}_{i=1}^{h_q}$, h_{kv} KV heads $\{\mathbf{K}_j, \mathbf{V}_j\}_{j=1}^{h_{kv}}$, with each head having the shape $n \times d_h$. The query heads are partitioned by a group size $G = h_q / h_{kv}$. The attention scores $\mathbf{S}_i$ and the attention output $\mathbf{O}_i$ for the i-th query head are computed by attending to its corresponding KV heads with the index $j = \lfloor (i-1)/G \rfloor + 1$:

$$\mathbf{S}_i = \text{Softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_j^\top}{\sqrt{d_h}}\right), \quad \mathbf{O}_i = \mathbf{S}_i \mathbf{V}_j. \tag{1}$$

The final output is obtained by concatenating the attention outputs and projecting them through a final linear layer $\mathbf{W}_O \in \mathbb{R}^{(h_q d_h) \times d}$: $\text{Attention}(\mathbf{X}) = \text{Concat}(\mathbf{O}_1, \ldots, \mathbf{O}_{h_q})\mathbf{W}_O$.

NSA. NSA (Yuan et al., 2025) is an enhancement of GQA designed for efficiency on long sequences. The key insight is that for long sequences, e.g., when $n > 32$k, the attention score matrix $\mathbf{S}$ exhibits strong sparsity. This allows for approximating the attention matrix by ignoring negligible values, leading to faster computation. As illustrated in Figure 2, NSA utilizes three distinct

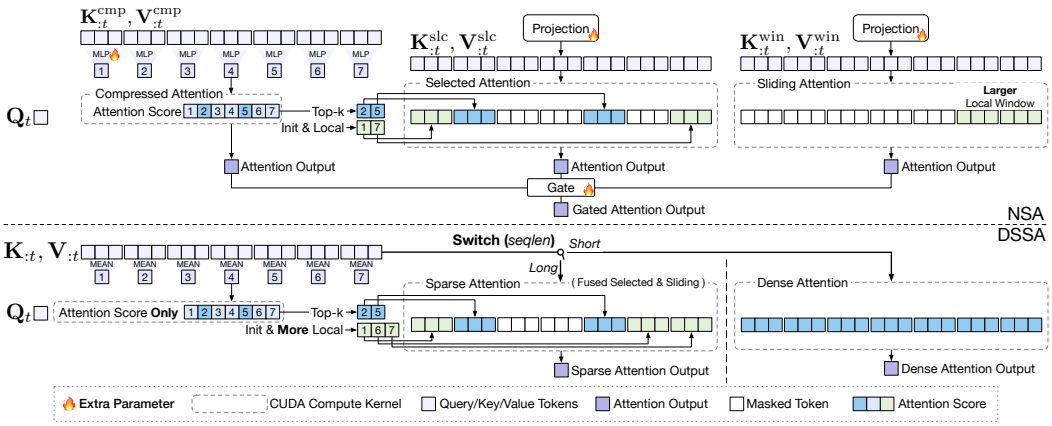

Figure 2: The overview of NSA and our DSSA. DSSA uses a shared KV for both Sparse Attention and Dense Attention. DSSA fuses Selected Attention and Sliding Attention and eliminates the output of Compressed Attention. In addition, DSSA introduces no extra parameters.

modules and combines them using a gating module. Based on the observation that adjacent attention scores are similar (Jiang et al., 2024), NSA splits the sequences into blocks of size B. First, *Compressed Attention* employs a compressed representation of the KV tensors to reduce the computational complexity. Second, *Selected Attention* leverages the attention scores from compressed attention to compute only the blocks with high attention scores. Finally, *Sliding Attention* is used to focus on local contextual information within the sequence. For these three attention modes, they introduce three sets of KV projection matrices: $\mathbf{W}_K^{\text{cmp}}, \mathbf{W}_V^{\text{cmp}}, \mathbf{W}_K^{\text{slc}}, \mathbf{W}_V^{\text{slc}}, \mathbf{W}_K^{\text{win}}, \mathbf{W}_V^{\text{win}}$. This final output can be mathematically represented as $\text{Output} = g^{\text{cmp}}\mathbf{O}^{\text{cmp}} + g^{\text{slc}}\mathbf{O}^{\text{slc}} + g^{\text{win}}\mathbf{O}^{\text{win}}$, where $\mathbf{O}^{\text{cmp}}$, $\mathbf{O}^{\text{slc}}$, and $\mathbf{O}^{\text{win}}$ are the outputs of the three respective modules, and the gate scores g^{cmp}, g^{slc}, and g^{win} are derived from the input features $\mathbf{X}$ via an MLP and a sigmoid activation. They also train an MLP module for compressing the KV tensors. The three distinct KV projections, combined with an additional MLP and gating module, result in a highly complex architecture. This complexity, in turn, makes the model poorly suited for training from scratch on short-sequence data and also complicates the process of converting pretrained dense models to sparse ones.

3.2 Overall Framework

We propose DSSA, a more concise framework with zero extra parameters that more closely aligns dense and sparse attention patterns.

Shared Key-Value Projection. We find that using three separate sets of KV projection parameters in NSA (Yuan et al., 2025) is unnecessary, which not only complicates the adaptation from short to long sequences but also significantly slows down computation for short sequences. Therefore, we propose using a single shared set of projection parameters, $\mathbf{W}_K$ and $\mathbf{W}_V$, initialized with the pretrained dense attention parameters and used for finetuning on long sequences.

Aligned Computation. In addition to ensuring that sparse and dense attention share the same parameters, their computational processes must also be closely aligned. In NSA, the three attention modules all generate outputs that are aggregated by an extra gating module. This forces the computation of all three modules even for short sequences, leading to substantial overhead. To mitigate this, we take a union of the two sparse patterns in *Selected Attention* and *Sliding Attention* and eliminate the output of *Compressed Attention*, forming a unified *Sparse Attention* module. Specifically, the original *Selected Attention* module identifies important token blocks based on the attention scores from the *Compressed Attention* module, $\mathbf{S}^{\text{cmp}}$. For a query token with index i, located in the block $b_i = \lfloor \frac{i-1}{B} \rfloor + 1$, attention is always granted to a fixed set of initial blocks and a set of local blocks:

$$\mathcal{I}_{\text{init}} = \{1, 2, \ldots, N_{\text{init}}\}, \quad \mathcal{I}_{\text{local}}(i) = \{b_i - N_{\text{local}} + 1, \ldots, b_i - 1, b_i\}. \tag{2}$$

The top-k selection is then applied to $\mathbf{S}^{\text{cmp}}$ over the set of remaining blocks, denoted as $\mathcal{I}_{\text{topk}}(i)$. The complete set of attended block indices for this query token is the union of these three sets:

$$\mathcal{I}(i) = \mathcal{I}_{\text{init}} \cup \mathcal{I}_{\text{local}}(i) \cup \mathcal{I}_{\text{topk}}(i). \tag{3}$$

If we denote the set of token indices in the j-th block as $T_j = \{jB+1, \ldots, (j+1)B\}$, the selected attention allows a token in the block b_i to attend to the union of blocks $\bigcup_{j \in \mathcal{I}(i)} T_j$. The *Sliding Attention*, on the other hand, allows the i-th token to attend to a range $\{i-w+1, \ldots, i\}$ of window size w. Since the local blocks in *Selected Attention* and the window in *Sliding Attention* create overlapping, we merge them by expanding the number of local blocks within our unified *Sparse Attention* to strictly cover the region of the *Sliding Attention*, that is, $N_{local} \geq \lceil \frac{w}{B} \rceil + 1$, as illustrated in Figure 3.

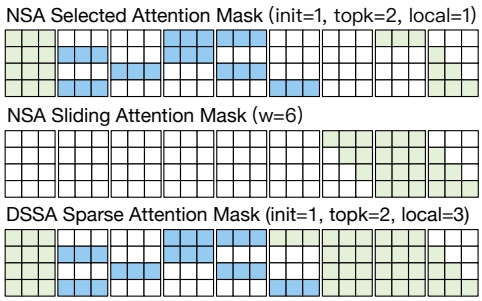

Figure 3: The illustration of the union of *Selected Attention* and *Sliding Attention*.

Furthermore, we eliminate the output of the *Compressed Attention* module, only retaining its attention scores $\mathbf{S}^{\text{cmp}}$ for block selection in *Sparse Attention*. This single-output design more closely mirrors dense attention and aids the training of the sparse attention model. DSSA can thus dynamically switch between dense and sparse attention patterns based on the input sequence length.

Simplified and Efficient Compression Module. Since we eliminate the output of the *Compression Attention*, using MLP for token compression would not receive gradients. We replace it with a more intuitive parameter-free pooling function, which will be detailed in Section 3.3. Additionally, computing the attention scores $\mathbf{S}^{\text{cmp}}$ introduces non-negligible overhead, and we will reduce this overhead in Section 3.4.

3.3 Block Representation

Simply compressing a long sequence with a large block size B in 1-stage can lead to a significant loss of granular information (Yuan et al., 2025). To address this, we implement a 3-stage, coarse-grained to fine-grained compression process, as shown in Figure 4. In the first stage, we process the input key sequence $\mathbf{K}$ to produce an intermediate and coarse-grained representation $\mathbf{K}^{C_1}$. By denoting the initial compression block size as l_{C_1} and the stride as s_{C_1}, we achieve this by applying a **mean-pooling** operation over sequential blocks:

$$\mathbf{K}_i^{C_1} = \text{Mean}(\mathbf{K}_{i \cdot s_{C_1} : i \cdot s_{C_1} + l_{C_1}}). \quad (4)$$

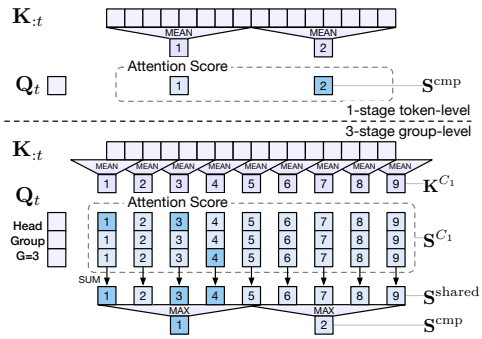

Figure 4: The illustration of the 3-stage group-level compression, compared with the 1-stage token-level compression.

Then, we compute the attention scores $\mathbf{S}^{C_1}$ between the query $\mathbf{Q}$ and $\mathbf{K}^{C_1}$:

$$\mathbf{S}^{C_1} = \text{Softmax}(\mathbf{Q}(\mathbf{K}^{C_1})^\top). \quad (5)$$

In the second stage, we employ block-wise sparse attention rather than token-level approaches for the efficiency of *Sparse Attention*. In a model utilizing GQA, we can achieve this by forcing the block selection pattern across all heads within a group to be the same. We conduct **summation** within the head group to get the shared importance score $\mathbf{S}^{\text{shared}}$:

$$\mathbf{S}^{\text{shared}} = \sum_{h=1}^{G} \mathbf{S}^{C_1}(h). \quad (6)$$

In the third stage, we apply a **max-pooling** operation, which can preserve the most salient features. The aggregated score $\mathbf{S}^{\text{cmp}}$ are defined as follows and used for the *Sparse Attention*:

$$\mathbf{S}_i^{\text{cmp}} = \text{Max}(\mathbf{S}^{\text{shared}}_{i:s:i \cdot s+l}). \quad (7)$$

In our method, we set $l_{C_1} = \frac{B}{2}$, $s_{C_1} = \frac{B}{4}$, $l = 5$, and $s = 4$ so that it can achieve the same compression ratio as 1-stage compression of block size B. Intuitively, we compute the sparse scores of the entire block based on several sliding sub-blocks within the block.

Algorithm 1 Computation of $\mathbf{S}^{\text{shared}}$ (Suppose $h_{kv} = 1$ without loss of generality.)

Require: $\mathbf{Q} \in \mathbb{R}^{n \times G \times d_h}, \mathbf{K}^{C_1} \in \mathbb{R}^{(n/s_{C_1}) \times d_h}, \mathbf{K}^{C_2} \in \mathbb{R}^{(n/s_{C_2}) \times d_h}$ in HBM. Block sizes B_q, B_k.
Divide $\mathbf{Q}$ into $T_q = \lceil n/B_q \rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_q}$ of size $B_q \times G \times d_h$ each.
Divide $\mathbf{K}^{C_1}$ into $T_1 = \lceil n/s_{C_1}/B_k \rceil$ blocks $\mathbf{K}^{C_1}_1, \ldots, \mathbf{K}^{C_1}_{T_1}$ of size $B_k \times d_h$ each.
Divide $\mathbf{K}^{C_2}$ into $T_2 = \lceil n/s_{C_2}/B_k \rceil$ blocks $\mathbf{K}^{C_2}_1, \ldots, \mathbf{K}^{C_2}_{T_2}$ of size $B_k \times d_h$ each.
Divide $\mathbf{S}^{\text{shared}}$ into $T_q \times T_1$ blocks of size $B_q \times B_k$ each.
for $i = 1, \ldots, T_q$ (parallel) **do**
  Load $\mathbf{Q}_i$ from HBM to on-chip SRAM.
  On chip, initialize online-softmax related statistic log-sum-exp lse.
  for $j = 1, \ldots, T_2$ (sequential) **do** ▷ First pass (Coarse-grained)
    Load $\mathbf{K}^{C_2}_j$ from HBM to on-chip SRAM.
    On chip, compute attention scores $\mathbf{S}^{C_2}_{ij} \in \mathbb{R}^{G \times B_q \times B_k}$ as in Eq. (8) and update lse.
  for $j = 1, \ldots, T_1$ (sequential) **do** ▷ Second pass (Fine-grained)
    Load $\mathbf{K}^{C_1}_j$ from HBM to on-chip SRAM.
    On chip, compute attention scores $\mathbf{S}^{C_1}_{ij} \in \mathbb{R}^{G \times B_q \times B_k}$ as in Eq. (5) and normalize it using lse.
    On chip, compute the final block $\mathbf{S}^{\text{shared}}_{ij} \in \mathbb{R}^{B_q \times B_k}$ by summing $\mathbf{S}^{C_1}_{ij}$ over the head group.
    Write the block $\mathbf{S}^{\text{shared}}_{ij}$ to its corresponding position in HBM.
return the output $\mathbf{S}^{\text{shared}}$.

3.4 Efficient Implementation

For efficient *Sparse Attention*, we follow the techniques in NSA (Yuan et al., 2025) to set the group size G of GQA to 16, a configuration well-suited for block sparse attention. More details can be found in Appendix B. **However, our profiling reveals that the computation of the compression score, $\mathbf{S}^{\text{cmp}}$, introduces a significant performance bottleneck.** A primary source of this slowdown is the substantial I/O required to store the first-stage attention scores $\mathbf{S}^{C_1}$ into the slow GPU HBM. The amount of data that needs to be written is $h_q n^2 / s_{C_1}$, where n is the full sequence length. Given that $s_{C_1} \ll n$, materializing the full attention score matrix to GPU HBM incurs a prohibitive cost.

Drawing inspiration from FlashAttention (Dao, 2024), we aim to minimize this I/O by ensuring the attention scores remain within the fast GPU SRAM as much as possible. Our approach, ***Fused Head Group Summation***, is to fuse the summation over the head group, required for the second-stage compression, directly into the SRAM-based computation loop of FlashAttention. After that, we can only store the reduced attention scores $\mathbf{S}^{\text{shared}}$ into GPU HBM, whose size is $h_q n^2 / (s_{C_1} G)$.

Another challenge arises from the fact that summing over the head group dimension and performing the online-softmax (Dao, 2024) along the sequence dimension are not commutative operations. This conflict prevents a straightforward fusion. To overcome this, we implement a two-pass approach. In the first pass, we compute the log-sum-exp (lse) term required for the softmax normalization within the SRAM. In the second pass, we leverage the lse to calculate the final attention scores, perform the summation across the head group within the SRAM, and write the reduced scores to the HBM. The trade-off of this two-pass method is that it doubles the computational workload. Therefore, we propose ***LSE Approximation*** to approximate the lse computation by using a coarser-grained attention score $\mathbf{S}^{C_2}$. Following Eq. (4) and Eq. (5), we change them to

$$\mathbf{K}^{C_2}_i = \text{Mean}(\mathbf{K}_{i \cdot s_{C_2} : i \cdot s_{C_2} + l_{C_2}}), \quad \mathbf{S}^{C_2} = \text{Softmax}(\mathbf{Q}(\mathbf{K}^{C_2})^\top). \tag{8}$$

By setting $s_{C_2} = 4 s_{C_1}$ and $l_{C_2} = 4 l_{C_1}$, the computational overhead was reduced from $2\times$ to $1.25\times$. We summarize the procedure for computing $\mathbf{S}^{\text{shared}}$ in Algorithm 1. To further reduce memory I/O, the max-pooling and top-k operations related to $\mathbf{S}^{\text{cmp}}$ could also be fused into the kernel; however, we leave this implementation for future work.

4 Experiment

We evaluate DSSA on tasks ranging from short to long contexts, and demonstrate its efficiency.

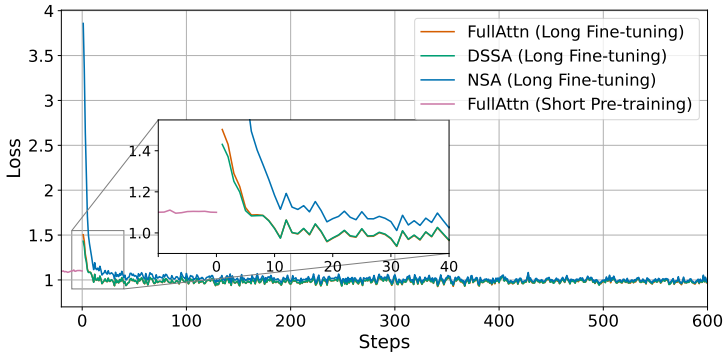

Figure 5: The training loss of models. We only show the last few iterations of the short pretraining.

4.1 Experiment Setup

Pretraining Setup. We first use full attention to pretrain a model on short-sequence data, marked as Short. We employ a standard GQA (Ainslie et al., 2023) model backbone with 8B parameters, with the hidden size $d = 4096$, the number of heads $h_q = 32$, $h_{kv} = 2$, and the head dimension $d_h = 128$. The pretraining dataset consists of 8T tokens of 4k-length sequences, primarily comprising FineWeb-Edu (Penedo et al., 2024) and Stack-v2 (Lozhkov et al., 2024). We set 8M tokens per batch, and use a WSD learning rate scheduler (Hu et al., 2024) with 2000 warmup steps to an initial learning rate of 7.5e-3, and 27000 decay steps to the final learning rate of 3e-4.

Long-Context Adaptation. When transitioning to long-context finetuning, we switch to DSSA-Sparse mode. Following NSA (Yuan et al., 2025), we set the compression block size $l_{C_1} = 32$, compression stride $s_{C_1} = 16$, and attention block size $B = 64$. For our efficient block selection implementation in Section 3.4, we additionally set the LSE Approximation block size $l_{C_2} = 128$ and stride $s_{C_2} = 64$. We set the selected block count $|\mathcal{I}| = 96$ (including $|\mathcal{I}_{\text{init}}| = 1$, $|\mathcal{I}_{\text{topk}}| = 63$, and $|\mathcal{I}_{\text{local}}| = 32$) for both training and inference. Therefore, the total number of visible tokens is $|\mathcal{I}| \cdot B = 6$k. We conduct long-sequence finetuning on the pretrained model using 5B tokens, with an initial learning rate of 3e-4 and linear decay to 2.75e-4. The training batches contain sequences from four length intervals: 0-4k, 4-12k, 12-24k, and 24-32k, with token counts in a 1:1:1:1 ratio.

Baselines. We finetune a baseline model with full attention, marked as FullAttn, using the same training configuration as DSSA-Sparse. We then apply several typical training-free sparse attention methods on FullAttn as baselines, including InfLLM (Xiao et al., 2024a) and MInference (Jiang et al., 2024). In addition, we present the results of Short with YaRN (Peng et al., 2023) to extend the context window size. In terms of trainable sparse attention, we compare with NSA (Yuan et al., 2025). By using the same training settings as in DSSA-Sparse, we finetune our pretrained model into an NSA version. We initialize NSA's three sets of KV parameters by replicating the original KV parameters in dense attention. As NSA does not publish their code, we adopt an open-source Triton implementation of NSA for experiments[1].

For all the above sparse attention methods, we maintain the same sparsity level to ensure a fair comparison. We provide the training curve for trainable methods in Figure 5. NSA causes a disruption in the loss, while DSSA is closer to FullAttn.

4.2 Task Performance

In this section, we evaluate DSSA and other baselines across various tasks. Notably, while the original NSA paper demonstrates performance comparable to full attention when training on long sequences from scratch, NSA fails to achieve satisfactory results in short-to-long adaptation settings. *This indicates that the substantial parameter overhead introduced by* NSA *renders it unsuitable for the conventional "pretraining-on-short, finetuning-on-long" paradigm.*

Long-Context Understanding. To evaluate DSSA's performance on long-input tasks, we compare DSSA and different baselines on RULER (Hsieh et al., 2024), LongBench (Bai et al., 2024) and LongPPL (Fang et al., 2025). RULER is a synthetic dataset with a configurable average length.

[1] https://github.com/XunhaoLai/native-sparse-attention-triton

Table 1: Task Performance on RULER. Best results in sparse attention are bolded.

Method	SG1	SG2	SG3	MK1	MK2	MK3	MV	MQ	VT	CWE	FWE	QA1	QA2	Avg. ↑
FULLATTN	100.00	100.00	100.00	96.00	94.00	92.00	82.00	98.50	93.20	44.40	91.33	48.00	56.00	84.26
SHORT+YARN	98.00	68.00	50.00	46.00	6.00	0.00	32.00	31.50	36.00	21.40	87.33	26.00	26.00	40.63
INFLLM	98.00	6.00	4.00	10.00	10.00	10.00	9.00	7.50	70.00	16.00	80.67	18.00	24.00	27.94
MINFERENCE	**100.00**	**100.00**	**100.00**	76.00	36.00	46.00	79.50	93.50	88.00	**64.20**	**92.67**	32.00	**44.00**	73.22
NSA	**100.00**	88.00	82.00	54.00	38.00	30.00	59.00	61.50	56.00	34.40	86.00	56.00	34.00	59.92
DSSA-SPARSE														
w/ *LSE Approx*	**100.00**	**100.00**	**100.00**	**94.00**	**82.00**	62.00	**98.50**	94.50	**98.00**	50.40	82.67	**72.00**	40.00	**82.62**
w/o *LSE Approx*	**100.00**	**100.00**	**100.00**	92.00	80.00	**64.00**	**98.50**	**95.50**	**98.00**	47.80	81.33	70.00	40.00	82.09
DSSA-DENSE	100.00	100.00	100.00	94.00	98.00	98.00	99.00	98.00	98.40	52.80	90.00	76.00	44.00	88.32

Table 2: Task Performance on LongBench and LongPPL. Best results in sparse attention are bolded.

Benchmark	FULLATTN	SHORT+YARN	INFLLM	MINFERENCE	NSA	DSSA-SPARSE	DSSA-DENSE
LongBench ↑	42.30	37.86	32.30	41.55	37.10	**42.54**	42.49
LongPPL ↓	2.06	5.28	12.01	2.62	4.24	**2.12**	2.00

LongBench is a bilingual benchmark for long-context understanding. Compared to RULER, LongBench is primarily built from existing, real-world datasets. LongPPL is a perplexity evaluation benchmark for long sequences. The experimental results of RULER when the length is 32k are shown in Table 1. The results on LongBench and LongPPL are shown in Table 2. Please refer to Appendix C for detailed performance of the sub-tasks in LongBench. From the results, we can observe that: 1) DSSA achieves the best performance compared to other sparse methods, with its results being highly competitive and closely matching the strong, FULLATTN baseline. Alternative approaches, whether applying training-free sparsity or training-based sparsity, result in a substantial drop in performance. 2) Compared to NSA, DSSA can achieve significant performance improvements through minimal finetuning on long-sequences. Although NSA has low training loss, its high perplexity on the LongPPL evaluations indicates that NSA has not adequately learned long-range dependencies. 3) A unique advantage of DSSA is the flexibility to seamlessly switch between dense mode and sparse mode. This flexibility not only provides an option for dense computation but can also lead to a further improvement in performance, surpassing even the full attention baseline. 4) Furthermore, the DSSA-SPARSE variant with LSE Approximation does not lose any performance, confirming the effectiveness of our acceleration technique. Given its comparable performance and efficiency, we adopt the LSE Approximation setting for all subsequent tasks by default.

Long Reasoning. To evaluate the performance of DSSA in long-output scenarios, we compared several major Long Reasoning tasks, including MATH-500 (Hendrycks et al., 2021b), AIME (MAA), and LiveCodeBench (LCB) (Jain et al., 2025). We finetune DSSA and baselines on OpenMathReasoning (Moshkov et al., 2025) and OpenCodeReasoning (Ahmad et al., 2025). As InfLLM and MInference primarily accelerate long-input processing, we exclude them from this long-output evaluation. The experimental results are shown in Table 3. The results show that DSSA attains performance on par with full attention, confirming its effectiveness for long-output scenarios.

Table 3: Task Performance on Long Reasoning Tasks.

Method	MATH-500	AIME 24	AIME 25	LCB v5	LCB v6	Avg. ↑
FULLATTN	86.00	37.50	30.63	30.67	29.14	42.79
NSA	83.80	28.75	23.54	25.15	25.14	37.28
DSSA-Sparse	87.80	38.33	29.38	29.94	27.83	42.66
DSSA-Dense	86.40	36.67	23.33	29.94	26.29	40.53

General Tasks. We verify that the DSSA architecture can freely switch back to Dense mode without performance degradation on short-sequence tasks after long-sequence fine-tuning. Zero-shot evaluations on MMLU (Hendrycks et al., 2021a), MMLU-Redux (Gema et al., 2025), CEval (Huang et al., 2023), MATH-500 (Hendrycks et al., 2021b), HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021) and BBH (Suzgun et al., 2023) are shown in Table 4. Experimental results show that DSSA achieves performance comparable to full attention.

Table 4: Task Performance on General Tasks.

Method	MMLU	MMLU-Redux	CEval	MATH-500	HumanEval	MBPP	BBH	Avg. ↑
SHORT	72.73	72.71	76.17	54.40	70.73	75.49	51.90	67.73
FULLATTN	73.38	70.24	78.11	54.60	71.34	75.10	49.13	67.41
NSA	68.27	66.39	74.33	44.40	62.20	65.00	43.81	60.63
DSSA-Dense	71.29	69.73	77.70	54.80	73.17	73.54	47.09	66.76

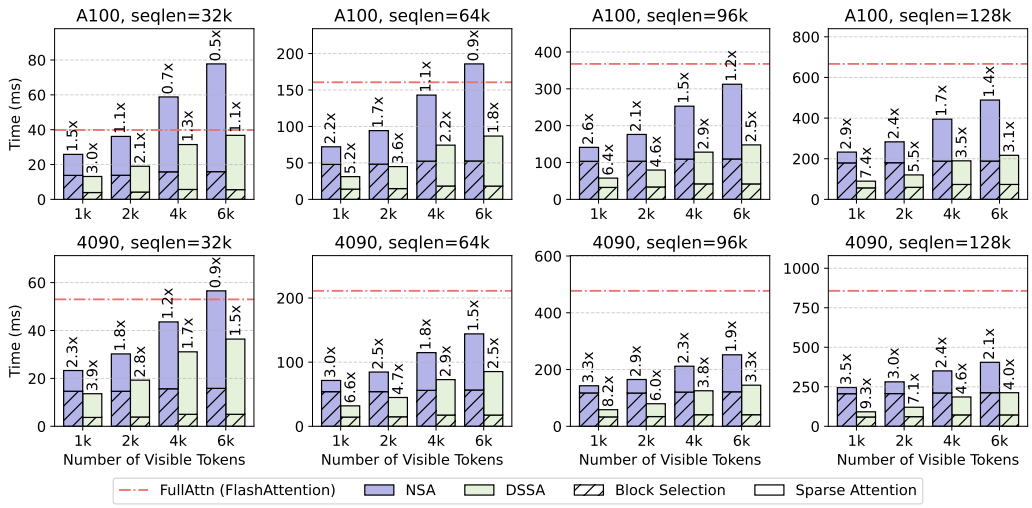

Figure 6: Speed of the kernels on NVIDIA A100 and NVIDIA 4090. "Number of Visible Tokens" means the number of key/value tokens each query token can attend to.

Table 5: Ablation study of *Block Selection* efficiency, with and without *LSE Approximation*. All measurements are in time (ms), and the number of selected blocks is set to 16.

Device	A100				4090			
Sequence Length	32k	64k	96k	128k	32k	64k	96k	128k
w/o LSE Approximation	4.67	18.20	42.46	75.36	4.89	19.95	46.51	83.26
w/ LSE Approximation	**3.93**	**14.07**	**32.44**	**56.59**	**3.70**	**14.39**	**33.16**	**59.04**

4.3 Efficiency

We first evaluate the efficiency of our kernel implementation on NVIDIA A100 and NVIDIA 4090. We evaluate DSSA's inference efficiency on the batch=1 setting. We select FlashAttention-2 (Dao, 2024) implementation for full attention. For a fair efficiency comparison with NSA, we ignore its sliding attention component, and compare solely on the compression and sparse attention parts by selecting an equal number of blocks $|\mathcal{I}|$. Experiment results are shown in Figure 6. When the number of selected blocks is 16, DSSA achieves up to $7.4\times$ over FlashAttention on A100 and $9.3\times$ on 4090. In contrast, NSA's speedup is limited to $3.5\times$ in the same setting. The breakdown of the execution time shows that the overhead from the *Block Selection* stage is greatly optimized by our efficient implementation in Section 3.4. We further conduct an ablation study on the *Block Selection*, as shown in Table 5, which shows the effectiveness of our proposed *LSE Approximation*.

The end-to-end inference speed (with a $|\mathcal{I}| = 96$ and W4A16 quantization (Frantar et al., 2025)) is shown in Figure 7. DSSA can achieve $2.13\times$ prefilling speedup and $2.32\times$ decoding speedup. Since DSSA does not accelerate the Feed-Forward Network (FFN) layers, a higher speedup ratio can be achieved by incorporating FFN-specific acceleration techniques in future work.

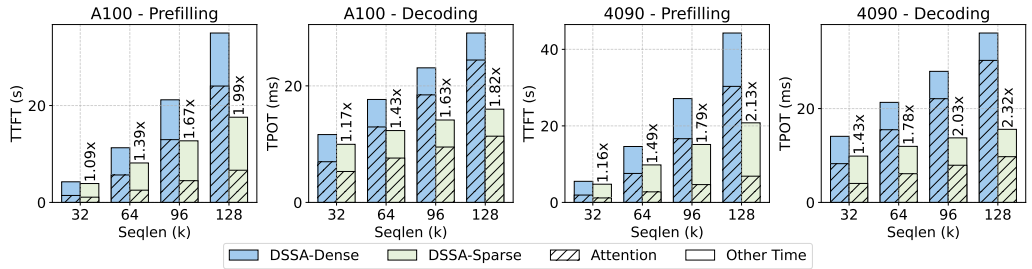

Figure 7: End-to-end inference speed of our 8B model when the number of visible tokens is 6k. TTFT means time-to-first-token, and TPOT means time-per-output-token.

5 Conclusion

In this paper, we introduced DSSA, a Dense-Sparse Switchable Attention framework designed to overcome the limitations of existing trainable sparse attention mechanisms. By ensuring architectural alignment with the standard pretrain-on-short and finetune-on-long workflow, DSSA facilitates a seamless and efficient sparse adaptation to long contexts without requiring extra parameters or causing disruptive distributional shifts. We believe DSSA offers a practical and powerful solution for advancing the capabilities of large language models in the long-context era.

Ethics statement

Our method has no potential risk since we focus on developing a novel algorithm.

Reproducibility statement

We provided our kernel implementation in the supplementary materials.

The Use of Large Language Models

After writing the paper manually, the paper was polished for clarity and readability using a large language model (LLM).

References

Wasi Uddin Ahmad, Sean Narenthiran, Somshubra Majumdar, Aleksander Ficek, Siddhartha Jain, Jocelyn Huang, Vahid Noroozi, and Boris Ginsburg. Opencodereasoning: Advancing data distillation for competitive coding. *arXiv preprint arXiv:2504.01943*, 2025.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024.

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

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *Preprint*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Guoxuan Chen, Han Shi, Jiawei Li, Yihang Gao, Xiaozhe Ren, Yimeng Chen, Xin Jiang, Zhenguo Li, Weiyang Liu, and Chao Huang. Sepllm: Accelerate large language models by compressing one segment into one separator. In *Forty-second International Conference on Machine Learning*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *The Twelfth International Conference on Learning Representations*, 2024.

Daya DeepSeek, Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Lizhe Fang, Yifei Wang, Zhaoyang Liu, Chenheng Zhang, Stefanie Jegelka, Jinyang Gao, Bolin Ding, and Yisen Wang. What is wrong with perplexity for long-context language modeling? In *The Thirteenth International Conference on Learning Representations*, 2025.

Elias Frantar, Roberto L Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language models. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, pp. 239–251, 2025.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, et al. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, et al. Are we done with mmlu? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5069–5096, 2025.

Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. Lm-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3991–4008, 2024.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021b.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Shengding Hu, Yuge Tu, Xu Han, Ganqu Cui, Chaoqun He, Weilin Zhao, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. In *First Conference on Language Modeling*, 2024.

Yuxiang Huang, Binhang Yuan, Xu Han, Chaojun Xiao, and Zhiyuan Liu. Locret: Enhancing eviction in long-context llm inference with trained retaining heads on consumer-grade devices. *arXiv preprint arXiv:2410.01805*, 2024.

Yuxiang Huang, Mingye Li, Xu Han, Chaojun Xiao, Weilin Zhao, Ao Sun, Hao Zhou, Jie Zhou, Zhiyuan Liu, and Maosong Sun. APB: Accelerating distributed long-context inference by passing compressed context blocks across GPUs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10708–10727, 2025.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems*, 36: 62991–63010, 2023.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2023.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

MAA. American invitational mathematics examination-aime. URL `https://maa.org/maa-invitational-competitions/`.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *arXiv preprint arXiv:2504.16891*, 2025.

OpenAI. GPT-4 technical report. *Preprint*, 2023.

Aaron OpenAI, Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

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

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

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

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6), 2022. ISSN 0360-0300. doi: 10.1145/3530811.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

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

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024b.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Shang Yang, Haotian Tang, Yao Fu, Song Han, et al. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *The Thirteenth International Conference on Learning Representations*, 2025.

Renjun Xu and Jingwen Peng. A comprehensive survey of deep research: Systems, methodologies, and applications. *arXiv preprint arXiv:2506.12594*, 2025.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. In *Forty-second International Conference on Machine Learning*, 2025.

John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, YX Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Jintao Zhang, Rundong Su, Chunyu Liu, Jia Wei, Ziteng Wang, Pengle Zhang, Haoxu Wang, Huiqiang Jiang, Haofeng Huang, Chendong Xiang, et al. A survey of efficient attention methods: Hardware-efficient, sparse, compact, and linear attention. 2025a.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Haocheng Xi, Jun Zhu, Jianfei Chen, et al. Spargeattention: Accurate and training-free sparse attention accelerating any model inference. In *Forty-second International Conference on Machine Learning*, 2025b.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

A THE USE OF LARGE LANGUAGE MODELS

After writing the paper manually, the paper was polished for clarity and readability using a large language model (LLM).

B IMPLEMENTATION DETAIL

We have shown the implementation of *Block Selection* in Section 3.4. We show the implementation detail of *Sparse Attention* here in Algorithm 2.

Algorithm 2 Computation of *Sparse Attention*. (Suppose $h_{kv} = 1$ without loss of generality.)

Require: $\mathbf{Q} \in \mathbb{R}^{n \times G \times d_h}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d_h}$. Block sizes B_k.
Divide $\mathbf{Q}$ into n blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_n$ of size $G \times d_h$ each.
Divide $\mathbf{K}, \mathbf{V}$ into $T_k = \lceil n/B_k \rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_k}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_k}$ of size $B_k \times d_h$ each.
Divide $\mathbf{O} \in \mathbb{R}^{n \times G \times d_h}$ into n blocks of size $G \times d_h$ each.
Divide the log-sum-exp lse into n blocks of size G each.
for $i = 1, \ldots, n$ (parallel) **do**
  Load $\mathbf{Q}_i$ from HBM to on-chip SRAM.
  On chip, initialize $\mathbf{O}_i^{(0)} = (\mathbf{0})_{G \times d_h}$, $\ell_i^{(0)} = (\mathbf{0})_G$, $m_i^{(0)} = (-\infty)_G$.
  for $j = 1, \ldots, T_k$ (sequential) **do**
    if $\mathbf{K}_j$ in visible tokens (determined by the $|\mathcal{I}(i)|$ in Eq. 3) **then**
      Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
      On chip, compute attention scores $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^\top \in \mathbb{R}^{G \times B_k}$.
      On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \text{rowmax}(\mathbf{S}_{ij})) \in \mathbb{R}^G$.
      On chip, compute $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - m_i^{(j)}) \in \mathbb{R}^{G \times B_k}$.
      On chip, compute $\ell_i^{(j)} = \exp(m_i^{(j-1)} - m_i^{(j)}) \ell_i^{(j-1)} + \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^G$.
      On chip, compute $\mathbf{O}_i^{(j)} = \text{diag}(\exp(m_i^{(j-1)} - m_i^{(j)}))^{-1} \mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_{ij} \mathbf{V}_j$.
  On chip, compute $\mathbf{O}_i = \text{diag}(\ell_i^{(T_k)})^{-1} \mathbf{O}_i^{(T_k)}$.
  On chip, compute $lse_i = m_i^{(T_k)} + \log(\ell_i^{(T_k)})$.
  Write $\mathbf{O}_i$ to HBM as the i-th block of $\mathbf{O}$.
  Write lse_i to HBM as the i-th block of lse.
return the output $\mathbf{O}$ and the log-sum-exp lse.

Algorithm 3 Computation of *Dense Attention*. (Suppose $h_{kv} = 1$ without loss of generality.)

Require: $\mathbf{Q} \in \mathbb{R}^{n \times G \times d_h}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d_h}$. Block sizes B_q, B_k.
Divide $\mathbf{Q}$ into $T_q = G \times \lceil n/B_q \rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_q}$ of size $B_q \times d_h$ each.
Divide $\mathbf{K}, \mathbf{V}$ into $T_k = \lceil n/B_k \rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_k}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_k}$ of size $B_k \times d_h$ each.
Divide $\mathbf{O} \in \mathbb{R}^{n \times G \times d_h}$ into T_q blocks of size $B_q \times d_h$ each.
Divide the log-sum-exp lse into T_q blocks of size B_q each.
for $i = 1, \ldots, T_q$ (parallel) **do**
  Load $\mathbf{Q}_i$ from HBM to on-chip SRAM.
  On chip, initialize $\mathbf{O}_i^{(0)} = (\mathbf{0})_{B_q \times d_h}$, $\ell_i^{(0)} = (\mathbf{0})_{B_q}$, $m_i^{(0)} = (-\infty)_{B_q}$.
  for $j = 1, \ldots, T_k$ (sequential) **do**
    Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
    On chip, compute attention scores $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^\top \in \mathbb{R}^{B_q \times B_k}$.
    On chip, compute $m_i^{(j)} = \max(m_i^{(j-1)}, \text{rowmax}(\mathbf{S}_{ij})) \in \mathbb{R}^{B_q}$.
    On chip, compute $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - m_i^{(j)}) \in \mathbb{R}^{B_q \times B_k}$.
    On chip, compute $\ell_i^{(j)} = \exp(m_i^{(j-1)} - m_i^{(j)}) \ell_i^{(j-1)} + \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_q}$.
    On chip, compute $\mathbf{O}_i^{(j)} = \text{diag}(\exp(m_i^{(j-1)} - m_i^{(j)}))^{-1} \mathbf{O}_i^{(j-1)} + \tilde{\mathbf{P}}_{ij} \mathbf{V}_j$.
  On chip, compute $\mathbf{O}_i = \text{diag}(\ell_i^{(T_k)})^{-1} \mathbf{O}_i^{(T_k)}$.
  On chip, compute $lse_i = m_i^{(T_k)} + \log(\ell_i^{(T_k)})$.
  Write $\mathbf{O}_i$ to HBM as the i-th block of $\mathbf{O}$.
  Write lse_i to HBM as the i-th block of lse.
return the output $\mathbf{O}$ and the log-sum-exp lse.

Similar to FlashAttention (Dao, 2024), the algorithm divides the input into blocks. The differences are: 1) The FlashAttention block size B_k of $\mathbf{K}$, should divide the sparse attention block size B. That is, B should be a multiple of B_k. 2) The FlashAttention block of $\mathbf{Q}$ typically contains a single attention head and multiple tokens. We follow NSA (Yuan et al., 2025) to make it contain a group of attention heads of a single token, so that they can share the same sparse pattern. 3) The inner loop of the FlashAttention iterates over all blocks of $\mathbf{K}$, whereas our method's loop only covers the visible blocks of the sparse attention. We also show the FlashAttention implementation of Dense Attention to Algorithm 3 for reference.

C Benchmark Details

We provide the detailed results of the LongBench benchmark, mentioned in Table 2, in Table 6. Following LongBench (Bai et al., 2024), the "Overall" score is computed by the macro-average over the six task categories.

Table 6: Task Performance on LongBench. Best results in sparse attention are bolded.

Category		FullAttn (PT)	FullAttn (FT)	InfLLM	MInference	NSA	DSSA-Sparse	DSSA-Dense
Single-Doc QA	NarQA	18.17	21.38	**21.02**	20.16	18.34	20.75	21.03
	Qasper	30.98	43.80	34.92	44.51	39.96	**45.29**	45.29
	MFQA-en	43.81	55.07	49.39	**54.83**	51.35	53.53	53.54
	MFQA-zh	54.51	57.26	51.75	57.00	59.06	**59.33**	59.64
Multi-Doc QA	HotpotQA	48.49	50.13	44.03	48.00	46.78	**54.11**	54.07
	2WikiQA	32.71	39.54	30.58	36.22	35.33	**37.86**	37.86
	MuSiQue	23.22	24.68	17.85	**22.87**	16.97	21.74	21.24
	Dureader	33.00	33.54	33.01	**33.94**	33.62	33.39	33.29
Summary	GovReport	31.93	32.17	21.40	**32.21**	28.72	30.33	30.38
	QMSum	22.45	24.35	20.96	**25.05**	23.81	24.58	24.35
	MultiNews	26.46	26.70	22.90	**26.50**	25.02	25.71	25.75
	VCSUM	16.55	16.37	17.81	16.17	**19.12**	16.17	16.20
Few-shot Learning	TREC	65.50	45.00	**61.00**	43.50	23.50	22.50	24.00
	TriviaQA	85.67	84.35	75.78	81.93	83.95	**84.22**	84.22
	SAMSum	42.92	40.26	37.46	39.81	38.47	**40.69**	40.51
	LSHT	38.00	37.75	24.57	**35.75**	25.50	22.01	21.47
Synthetic Task	PsgCount	4.06	4.00	3.00	3.50	3.50	**5.00**	4.50
	PsgRe-en	20.75	86.50	19.00	85.00	66.00	**92.00**	91.00
	PsgRe-zh	42.00	90.50	43.00	90.50	68.00	**90.50**	90.50
Code	LCC	58.65	35.72	31.35	35.91	33.83	**44.73**	44.73
	RepoBen-P	43.93	35.00	30.72	34.17	34.95	**44.62**	44.76
Overall ↑		37.86	42.30	32.30	41.55	37.10	**42.54**	42.49

D Hyperparameters Analysis

In this section, we present analyses and discussions regarding the choice of key hyperparameters in our framework, specifically focusing on the number of selected blocks ($|\mathcal{I}_{\text{topk}}|$), local blocks ($|\mathcal{I}_{\text{local}}|$), initial blocks ($|\mathcal{I}_{\text{init}}|$), and the block size ($B$). These parameters serve as the core variables controlling the trade-off between model performance and computational efficiency. As shown in Table 7, increasing $|\mathcal{I}_{\text{topk}}|$ from 31 to 63 yields significant performance gains, whereas further increasing it to 95 offers only marginal improvements. Similarly, while larger $|\mathcal{I}_{\text{local}}|$ improves results, the marginal gain decreases as computational overhead grows. Consequently, we select $|\mathcal{I}_{\text{topk}}| = 63$ and $|\mathcal{I}_{\text{local}}| = 32$ as the optimal configuration for balancing efficiency and effectiveness.

Table 7: Hyperparameter analysis on the number of selected blocks and local blocks.

$\mathcal{I}_{\text{init}}$	$\mathcal{I}_{\text{topk}}$	$\mathcal{I}_{\text{local}}$	RULER ↑	LongPPL ↓
1	63	32	82.62	2.12
1	31	32	76.14	2.36
1	95	32	84.89	2.05
1	63	16	82.08	2.15
1	63	64	83.45	2.09

Regarding other parameters, we set $|\mathcal{I}_{\text{init}}|$ to a minimal value following the established practice in StreamingLLM (Xiao et al., 2024b), as a small number of attention sinks is sufficient for stability. For the block size B, we adopt $B = 64$ following NSA (Yuan et al., 2025). This choice is motivated by hardware efficiency, as FlashAttention kernels typically require block sizes of 64 or 128 to maximize Tensor Core utilization and memory bandwidth.