Efficient Sparse Attention needs Adaptive Token Release

Anonymous ACL submission

Abstract

In recent years, Large Language Models (LLMs) have demonstrated remarkable capabilities across a wide array of text-centric tasks. However, their 'large' scale introduces significant computational and storage challenges, particularly in managing the key-value states of the transformer, which limits their wider applicability. Therefore, we propose to adaptively release resources from caching and rebuild the necessary key-value states. Particularly, we accomplish this by a lighting controller module to approximate an ideal top-K sparse attention. This module retains the tokens with the highest top-K attention weights and simultaneously rebuilds the discarded tokens, which may become essential for future decoding. Comprehensive experiments in natural language generation and natural language modeling task reveal that our method is not only competitive with full attention in terms of performance but also achieves a significant throughput improvement of up to **221.8**%. The code for replication is available on the `https://anonymous.4open.science/r/ADORE-5384`.

1 Introduction

After breaking through the cognitive barriers, large language models (LLMs) are now widely used in many text-rich areas, such as voice assistants (Zhang et al., 2023), search engines (Lindemann, 2023), and recommendation systems (Acharya et al., 2023). These successes are a testament to the philosophy of scaling up parameters to boost performance, i.e., the scaling law (Kaplan et al., 2020). However, in situations demanding rapid or extensive text modeling, the vast size of the model significantly escalates the computational and storage requirements for the key-value (KV) states of self-attention, which, in turn, limits its throughput (Ma et al., 2023; Liu et al., 2023a). For example, when using a model with 7 billion parameters, caching the KV states for 1,000 tokens results in a memory requirement that exceeds twice the size of the model parameters, consequently increasing time costs in attention calculation and memory swapping.

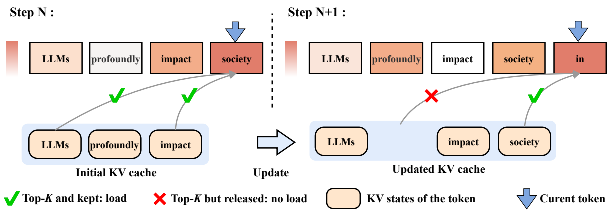


Figure 1: An illustration of the conflict of releasing Key-Value (KV) states in advance during the inference. Consider a cache size of 3. At step N, the KV states associated with the word 'profoundly' are released from the cache. Consequently, in the subsequent step N+1, the 'profoundly' state is absent from the cache, despite having a higher attention score for 'in'.

Recent efforts address this issue from two perspectives: 1) hardware optimization, analogous to 'increasing income'; 2) refining algorithms, similar to 'reducing expenditure'. The former approach typically optimizes performance by scheduling tasks across multiple GPUs, (Borzunov et al., 2022) or by implementing hierarchical unloading using the CPU and disk (Aminabadi et al., 2022; Sheng et al., 2023). These techniques, though efficient, require additional hardware and, if not carefully scheduled, can lead to increased communication latency. This, in turn, may potentially degrade the overall user experience (Rasley et al., 2020; Yang et al., 2023). The latter strategy enhances efficiency by limiting the caching size of key-value states, such as sparsely attending to its immediate neighbors (Zaheer et al., 2020; Beltagy et al., 2020) or compressing prompts (Jiang et al., 2023; Li et al., 2023d). Though efficient, it can often lead to a drop in performance. Besides, some methods instantiate the sparse attention by masking attention *after* the attention weights have been calculated (Rao et al., 2021; Li et al., 2023c).

As a consequence, they fail to enhance inference speed and reduce memory usage. Among these methods, the dynamic top-K attention (Liang et al., 2023), achieving sparsity by selecting the highest attention contributions, demonstrates performance comparable to, or even better than, full attention models. Due to its superior efficacy, it has been incorporated into numerous foundational architectures, including BiFormer (Zhu et al., 2023) and Informer (Zhou et al., 2021).

Despite the success of the dynamic top-K attention, it is non-trivial to simultaneously achieve high efficacy and efficiency. Firstly, to gain efficiency, releasing the unnecessary KV states of previous tokens in advance may result in inaccuracies of top-K attention calculation due to premature and erroneous releasing. This occurs because accurately determining the top-K attention requires considering all KV states of past tokens. Secondly, as illustrated in Figure 1, due to the long-term dependencies in text, the tokens released earlier could be among those needed for top-K attention in future decoding. As a consequence, its absence will lead to inaccurate sparse attention calculation for the later tokens.

To this end, we introduce ADORE, **AD**aptive t**O**ken **RE**lease, which maintains a constant cache size by accurately releasing useless past key-value (KV) states and efficiently reconstructing vital past KV states that were previously released. ADORE introduces a lightning controller module that adaptively releases tokens with the lowest predicted attention contribution for the current token from the KV cache. This ensures a fixed KV cache overhead, even when processing a large number of tokens. In addition, ADORE rebuilds the KV state for tokens that are likely to contribute higher attention scores but have been previously released. This rebuild mechanism counters the issue when a released token is essential for future decoding. Moreover, ADORE can seamlessly integrate into LLM inference, showing impressive results with only minor fine-tuning and training needed for the lightweight controller module. To demonstrate the effectiveness of our approach, we conducted extensive experiments on multiple benchmark datasets. The results reveal that ADORE achieves up to a 221.8% improvement in throughput compared to full attention models while preserving almost identical text quality.

2 Methodology

This section first establishes the framework for efficient sparse attention, followed by initially exploring the adaptive token release in Section 2.2. Subsequently, we rebuild the KV states of important tokens, approximating the ideal dynamic sparse attention in Section 2.3. Finally, we propose an optimized matrix slicing algorithm to accelerate the implementation of our method in Section 2.4. An overview of our method is illustrated in Figure 2.

2.1 Efficient Sparse Transformer

Let $T_n = \{t_1, \ldots, t_s, t_{s+1}, \ldots, t_n\}$ be a set of word tokens, where $\{t_1, \ldots, t_s\}$ represent user input tokens, and $\{t_{s+1}, \ldots, t_n\}$ are tokens generated by a transformer-based model, such as GPT-Neo (Black et al., 2021) and Llama (Touvron et al., 2023). When generating the next token t_{n+1}, the current token t_n serves as the query input. The t_n's key-value states are based on the following scaled dot-product attention as

$$\boldsymbol{a}_{n,l} = \mathrm{softmax}\left(\frac{\boldsymbol{q}_{n,l} \times (\boldsymbol{K}_l^n)^\top}{\sqrt{d}}\right) \times \boldsymbol{V}_l^n, \quad (1)$$

where $\boldsymbol{a}_{n,l} \in \mathbb{R}^d$ denotes the hidden state at the l^{th} layer of the transformer. It undergoes a non-linear transformation process to become the key and value states associated with the token t, $\boldsymbol{q}_{n,l}$ denotes the query vector derived from t_n at the l^{th} layer. The terms $\boldsymbol{K}_l^n \in \mathbb{R}^{(n)\times d}$ and $\boldsymbol{V}_l^n \in \mathbb{R}^{(n)\times d}$ represent the key and value states from the current token set T_n at the same layer. These states are retained in the GPU memory to minimize redundant computations. The generation of the token t_{n+1} is accomplished through a multi-classification approach, utilizing the hidden state $\boldsymbol{v}_{n,L} \in \mathbb{R}^d$ from the last layer.

For an efficient sparse transformer, we selectively cache the most relevant KV states, aiming to reduce computational demands while maintaining or even enhancing the model's performance in generating subsequent tokens as

$$\boldsymbol{a}'_{n,l} = \mathrm{softmax}\left(\frac{\boldsymbol{q}_{n,l} \times \left(\boldsymbol{K}_{m+1,l}^n\right)^\top}{\sqrt{d}}\right) \times \boldsymbol{V}_{m+1,l}^n.$$

Here, $\boldsymbol{a}'_{n,l}$ approximates the $\boldsymbol{a}_{n,l}$ using the $\boldsymbol{K}_{m+1,l}^n \in \mathbb{R}^{(m+1)\times d}$, $\boldsymbol{V}_{(m+1),l}^n \in \mathbb{R}^{(m+1)\times d}$, which correspond to selecting m rows from $\boldsymbol{K}_l^{n-1}$ and $\boldsymbol{V}_l^{n-1}$ and concatenating them with $\boldsymbol{k}_{n,l}$ and

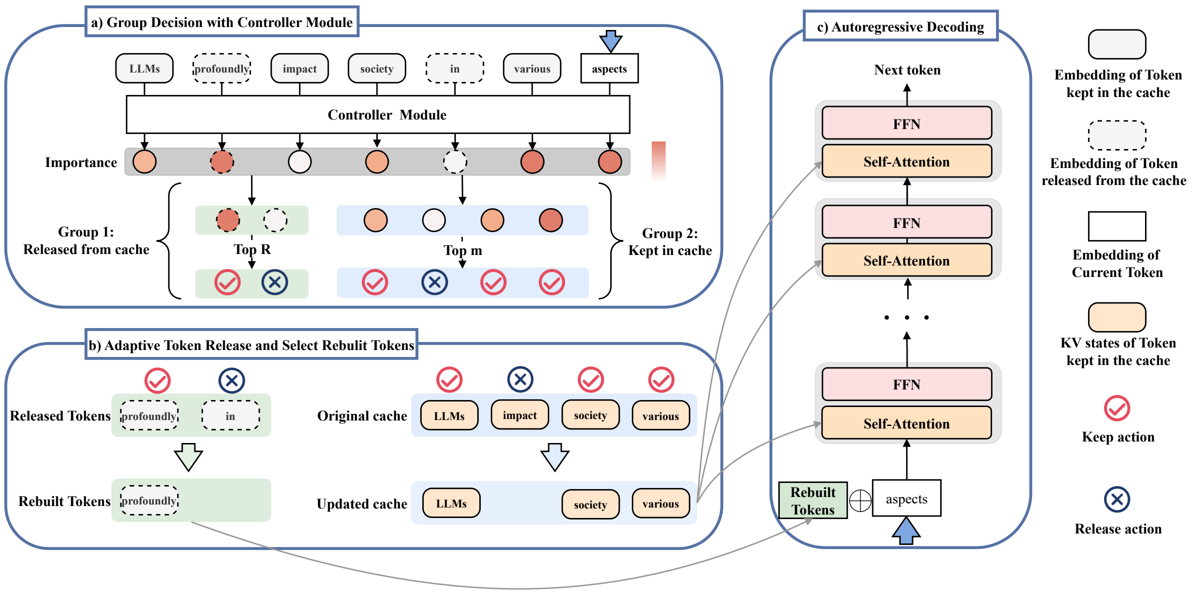


Figure 2: The controller module calculates the importance of all input and generated tokens for the current token. The Key-Value (KV) cache maintains the states of m tokens with the highest importance. For tokens that were previously released from the cache, those with the top-R highest importance are concurrently modeled alongside the current token.

$\boldsymbol{v}_{n,l}$ respectively, with the condition that $m << n-1$. $\boldsymbol{k}_{n,l}$, $\boldsymbol{v}_{n,l}$ denote the key and value vector derived from t_n at the l^{th} layer. This implies that only a significantly smaller $\boldsymbol{K}^n_{m+1,l}$ and $\boldsymbol{V}^n_{m+1,l}$ are retained in GPU for rapid inference and save memory.

From a performance standpoint, achieving the ideal sparsity involves computing the full attention weight $\boldsymbol{w}_n = \boldsymbol{q}_{n,l} \times (\boldsymbol{K}^n_l)^\top \in \mathbb{R}^n$ and then selecting the top-m query-key product weights. Then, these weights serve as indices for slicing $\boldsymbol{V}^n_{m+1,l}$. While this method is optimal in performance, it does not confer any computational or memory savings as the process of computing full attention weights for all query-key pairs and then selecting the top weights is computationally intensive.

2.2 Adaptive Token Release

The adaptive token release is to create efficient scheduling of the key-value states within the GPU memory. The main idea is to use a lighting controller module as an alternative to computing full weight for slicing the full key-value states. To be both efficient and effective, we have implemented several design strategies:

- **Refine the model with top-K attention.** Compared to the full attention, Top-K could mitigate the impact of excluding partial KV states once the pertinent top-K KV states are included within the m cached KV states, which is consistent with the target defined in Equ (2). Therefore, we initially fine-tune the LLMs with top-K attention, which utilizes only the highest top-K attention weights while setting the remainder to 0. Remarkably, this approach yields performance that is on par with full attention models (Liu et al., 2022). To be efficient, the cache size m is slightly larger than K. As m decreases, the complexity of the scheduling process increases correspondingly.

- **Adopt a uniform scheduling policy for the retention or exclusion of KV states across various layers.** Constructing a layer-specific scheduling strategy would necessitate additional time to model each layer's input. Moreover, the initial layer is more pivotal for integrating value states; as we delve deeper into the layers, the hidden states become increasingly homogeneous (Wu et al., 2023). Additionally, it is observed that different layers often focus on a similar set of top-K attentions. The effectiveness of the uniform scheduling policy is elaborated in Appendix C.

- **Update the cached KV states by appending the latest KV state and selectively release an older one.** An intuitive idea is to store the KV states in the motherboard's memory as backup. However, due to bandwidth limitations between the GPU and motherboard, moving KV states in and out proves to be extremely slow, at times even slower than re-calculating the KV states (Aminabadi et al., 2022).

Consequently, when updating the cached KV states, we simply append the most recently computed KV states while removing a nonsignificant older one, thereby maintaining a constant size for the cache.

Adhering to these strategies, we develop a controller module that utilizes the lightweight and time-efficient GRU (Dey and Salem, 2017) for scheduling the cached KV states. Specifically, during the generation of token t_{n+1}, we establish the probability of whether caching the KV state of a token t_i as:

$$\begin{aligned} \boldsymbol{z}_i &= \mathrm{GRU}(\boldsymbol{x}_i, \boldsymbol{z}_{i-1}) \\ \sigma_i &= \mathrm{Sigmoid}(\mathrm{MLP}(\boldsymbol{p}_i + \boldsymbol{z}_i)) \end{aligned}$$

where $\boldsymbol{x}_i \in \mathbb{R}^d$ represents the token embedding from the LLMs. The GRU is a single-layer GRU (an unidirectional model with its effectiveness analyzed in Appendix D) that recurrently transforms this token embedding into a context-aware representation $\boldsymbol{z}_i \in \mathbb{R}^{d'}$. The term $\boldsymbol{p}_i \in \mathbb{R}^{d'}$ denotes the position embedding for the i^{th} token, which signifies the importance of token position in the scheduling model. During the update of the KV states, we discard those with the lowest σ_i values and append the most recent KV states to the cached states. To fine-tune its parameters, we construct a dataset by collecting word embeddings of each sequence as input. Then we construct corresponding labels by assigning a value of 1 to the indices of the top-K tokens that most frequently occur within the top-$K/2$ attention scores across all layers, and a value of 0 to all others.

2.3 KV States Rebuild

Adaptive token releasing facilitates the selective preservation of the most pertinent tokens, yet previously discarded tokens may become essential for future decoding due to the long-term dependencies in text. To counter this issue, we propose the rebuilding of KV states as a complement.

This method entails retrieving the top-R tokens with the highest σ_i values from the set of released tokens. Let $\boldsymbol{X}_R \in \mathbb{R}^{R\times d}$ represent the token embedding of selected released tokens. We concatenate $\boldsymbol{X}_R$ with $\boldsymbol{x}_n$, i.e., the embedding of current token t_n, forming the input $\boldsymbol{X}_{R+1} \in \mathbb{R}^{(R+1)\times d}$. After $(l-1)$-layers processing, we can obtain the query states $\boldsymbol{Q}^n_{R+1,l} \in \mathbb{R}^{(R+1)\times d}$, $\boldsymbol{K}^n_{m+R+1,l} \in \mathbb{R}^{(m+R+1)\times d}$ and $\boldsymbol{V}^n_{m+R+1,l} \in \mathbb{R}^{(m+R+1)\times d}$, where $\boldsymbol{K}^n_{m+R+1,l}/\boldsymbol{V}^n_{m+R+1,l}$ is formulated by concating cached key/value state and rebuild key/value states for the input tokens. With its argument, the attention is calculated as

$$\boldsymbol{A}'_{R+1,l} = \mathrm{softmax}\left(\frac{\mathrm{Q}^{\mathrm{n}}_{\mathrm{R+1,l}} \times \left(\mathrm{K}^{\mathrm{n}}_{\mathrm{m+R+1,l}}\right)^\top}{\sqrt{\mathrm{d}}}\right) \times \mathrm{V}^{\mathrm{n}}_{\mathrm{m+R+1,l}},$$

where $\boldsymbol{A}'_{R+1,l}$ is the hidden state. To get the corresponding value for the current generating tokens, we get the $\boldsymbol{a}'_{n,l}$ by selecting the last row of $\boldsymbol{A}'_{R+1,l}$. Through the parallel rebuilding of the released KV states, we maximize the utilization of GPU without incurring excessive time overhead.

2.4 Matrix Slicing as Multiplication

The scheduling of KV states relies on the use of a matrix-slicing operator. Traditional slicing operators like `gather` and `mask-select` can lead to significant time overheads (Kim et al., 2022), particularly when batch operations involve varying slicing indices. To circumvent this, we leverage the GPU's rapid matrix multiplication capabilities. For instance, to remove the j^{th} row from $\boldsymbol{K}^n_{m,l}$, we initially prepare a slicing matrix, $S_j = I_{(1:j-1,j+1:m),:}$, where $I \in \mathbb{R}^{m\times m}$ is the identity matrix and $I_{(1:j-1,j+1:m),:}$ selects all rows of I except the j^{th} row. The resulting $\boldsymbol{K}^n_{m-1,l} = S_j \times \boldsymbol{K}^n_{m,l}$, with S_j being pre-prepared to save time.

3 Experiment

3.1 Datasets

To evaluate the effectiveness of various sparse attention mechanisms in LLM, we conduct extensive experiments across three distinct tasks: natural language generation, stream generation, and long-text modeling. For the first task, we evaluate on UltraChat (Ding et al., 2023), EverythingLM[1], and Math (Li et al., 2023b). For the second task, we experiment on StreamEval (Xiao et al., 2023) and StreamChat (built upon UltraChat). For the last task, we evaluate models on CNN Dailymail (See et al., 2017) and SAMSum (Gliwa et al., 2019).

Specifically, **UltraChat** is a multi-turn dialogue dataset containing approximately 696,600 training samples and covering diverse topics such as questions about the world and creative writing. **EverythingLM** is a instructional dataset consisting of 1,000 conversations and encompassing a wide array

[1] https://huggingface.co/datasets/totally-not-an-llm/EverythingLM-data

Dataset	UltraChat			EverythingLM			Math		
Metric	BLEU	ROUGE	BERT-F	BLEU	ROUGE	BERT-F	BLEU	ROUGE	BERT-F
Full Attention	35.6	29.2	63.4	35.4	30.8	64.5	38.6	29.9	69.7
Window Attention	26.7	28.0	61.4	22.3	25.9	62.3	30.3	24.3	66.3
Strided Attention	28.0	24.8	57.5	20.3	22.1	58.5	33.0	26.7	66.7
StreamingLLM	23.9	26.0	59.6	20.5	25.6	61.4	32.9	26.8	68.3
ADORE	**36.8***	**28.8**	**63.5***	**30.4***	**27.7***	**63.1***	**38.8***	**28.9***	**70.5***

Table 1: Performance comparison of different methods in natural language generation tasks. We use Full Attention as the upper limit. The best results are marked **bold**. "$*$" indicates significant improvement over the top-performing sparse attention method, with a p-value < 0.01.

of topics and interactions. **Math** dataset is composed of 50,000 problem-solution pairs obtained using GPT-4 across 25 math topics. **StreamChat** concatenates every 100 samples from UltraChat and feeds them into the model in a streaming fashion to assess the quality of the generated answers. **StreamEval** is a question-answer dataset with ground truth answers, building upon LongEval (Li et al., 2023a). Specifically, it comprises 2,000 samples, each with 1,000 lines of textual data and 100 retrieval questions. **CNN Dailymail** is a news summarization dataset containing over 300,000 news articles. **SAMSum** is a summarization dataset containing about 16,000 messenger-like conversations with summaries. The details of the datasets are reported in Appendix A.

Baseline. We compare our method with the following methods: **(1) Full Attention** encompasses all past KV states across every layer, characterized by a time complexity of $O(T^2)$ and linear growth in cache size. This method utilizes the most extensive token information, thus establishing an upper bound for most tasks. **(2) Window Attention** (Hassani et al., 2023) focuses on the nearest tokens for self-attention at each layer, thus ensuring a constant size for the key-value cache. **(3) Strided Attention** (Child et al., 2019) attends to both the nearest and distant tokens by periodically focusing on one with a fixed interval, thus striking a balance between effectiveness and efficiency. **(4) StreamingLLM** (Xiao et al., 2023) extends Window Attention by adding the first four tokens to the cache, aiming to maintain a normal distribution of attention scores and stable inference settings.

Experimental Protocols. We employ Llama-2 7B as our backbone for evaluation; it has 32 transformer layers and an extended 4,000 context length. The Llama-2 7B (Touvron et al., 2023) is known for its excellent performance and includes RoPE (Su et al., 2024) for simplified length extension. For our experiments, we employ the top-96 attentions and set the KV cache size m to 192 with top-8 re-

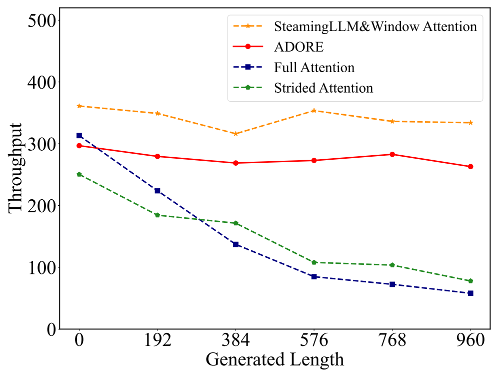


Figure 3: Performance comparison in terms of throughput for generating different text lengths.

built tokens. We randomly selected 1,000 samples from the benchmark dataset for training purposes. This sample was utilized to develop the sparse top-K backbone model using QLoRA (Dettmers et al., 2023), along with the controller module. The extra data were employed for testing models. To evaluate the quality of the generated text, we use metrics including BLEU, ROUGE, BERT-F (Zhang et al., 2019) and Accuracy. To measure the inference speed of different methods, we use **Throughput**, which is defined as the number of tokens generated per second.

3.2 Natural Language Generation

This subsection evaluates models' performance in natural language generation. We summarize the quality of generating text on UltraChat, EverythingLM and Math benchmarks in Table 1 and the throughput against different sequence lengths in Figure 3. From the results reported, we have the following observations:

- **The proposed ADORE achieves the best performance, and consistently outperforms all the baselines on all datasets**. In Table 1, our method shows an improvement over full attention in the UltraChat dataset, with increases of 1.2% in BLEU scores and 0.1% in BERT-F scores. On the other hand, Window Attention, Strided Attention, and

StreamingLLM show reductions of 13.1%, 15.1%, and 14.9% in BLEU scores, respectively. A similar trend is also observed in the learning curve illustrated in Appendix C.

- **Our proposal performs the best in achieving a high efficiency while maintaining a competitive performance against full attention**. Specifically, it is evident that our method demonstrates a consistent throughput against various generated text lengths; whereas full attention suffers from a significant drop on throughput as the generated text length increases. Notably, our method outperforms full attention by 151.4% and 221.8% when generating text lengths of 768 and 960, respectively. Though Window Attention and StreamingLLM have higher throughput, their performance on natural language generation suffers a lot.

3.3 Stream Generation

To show the real-world applicability of our proposal, we emulate the performance of the models on infinite streaming dialogue, i.e., StreamChat, and question-answering tasks, i.e., StreamEval. For StreamChat, we chunk the streaming chat with the size of 4096 to evaluate the quality of generation against different sequence lengths. The experimental results are reported in Table 2. For StreamEval, we report the generating accuracy of models' responses after multi-times query in Figure 4.

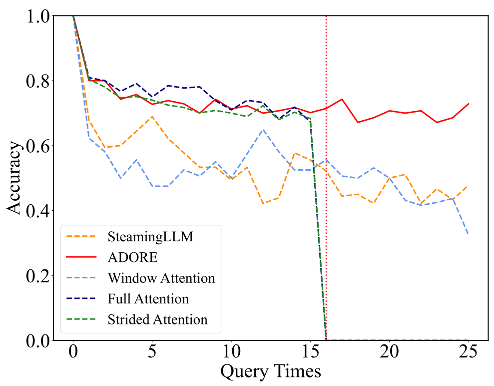


Figure 4: Performance comparison on the StreamEval at various query times.

From the Table 2 and Figure 4, we have following observations: **(1)** In the table, our method demonstrates a consistent performance across different sequence lengths, which justifies its efficacy in streaming dialogue, especially in length extrapolation and capturing high-importance tokens. While full attention exhibits the best performance on the first subset (length in range $(0, 4096]$), its performance rapidly declines as the streaming sequence length surpasses the pre-training window size, and eventually becomes almost 0. **(2)** In the figure, our method consistently maintains high accuracy, even when the number of queries exceeds 20, which expresses the superiority of our proposed method. On the other hand, full attention and strided attention display competitive performance at limited query times. However, they suffer a significant drop in performance due to Out-of-Memory (OOM) issues, which arise as the accumulation of excessive Key-Value (KV) states increases with the number of queries. This observation justifies the necessity of sparse attention. However, Window Attention and StreamingLLM demonstrate lower accuracy compared to our approach, primarily due to their fixed heuristic policies.

3.4 Long Text Modeling

We explore the performance of different methods in modeling super-long texts on CNN Dailymail and SAMSum. We report perplexity (ppl.) as the metric to compare the performance of different methods across different sequence length subsets. Similar to Section 3.3, the length in each subset is in the range of $((i-1) \times 1024, i \times 1024]$ for $(i = 1, 2, \ldots)$.

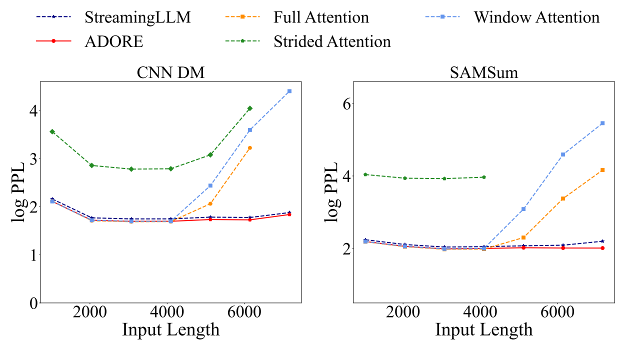


Figure 5: Perplexity evaluation on CNN DM and SAMsum across different lengths.

Figure 5 illustrates the logarithm of perplexity for different methods across various modeling intervals. It is evident that our method and StreamingLLM consistently maintain the lowest perplexity; they are effective in preserving the original attention distribution with sparse attention. Therefore, they both demonstrate superior performance on extrapolating length. Although full attention exhibits the best performance in the shortest input length subset ($[0, 4096]$), its performance

Sequence Length	(0, 4096]			(4096, 4096×2]			(4096×2, 4096×3]			(4096×3, 4096×4]		
Metrics	BLEU	ROUGE	BERT-F	BLEU	ROUGE	BERT-F	BLEU	ROUGE	BERT-F	BLEU	ROUGE	BERT-F
Full Attention	42.8	43.8	70.9	3.6	4.9	33.1	2.1	2.4	30.1	2.0	2.4	30.0
Strided Attention	27.7	30.5	60.9	2.1	3.0	29.7	2.0	2.2	30.0	2.0	2.1	29.6
Window Attention	24.7	28.5	60.6	14.2	19.6	54.3	16.1	18.1	50.1	19.9	20.4	52.1
StreamingLLM	14.6	36.1	64.8	14.8	29.0	63.2	18.6	28.4	62.4	21.7	27.5	63.5
Ours	**38.9**	**38.3**	**66.4**	**36.5**	**39.2**	**67.7**	**35.5**	**37.7**	**67.1**	**36.7**	**39.5**	**69.1**

Table 2: Performance comparison on StreamChat across different streaming lengths. The best results are shown in **bold**.

quickly becomes worse when the input length surpasses the size of the pretraining window.

3.5 Ablation Study

3.5.1 Influence of Attention Sparisity

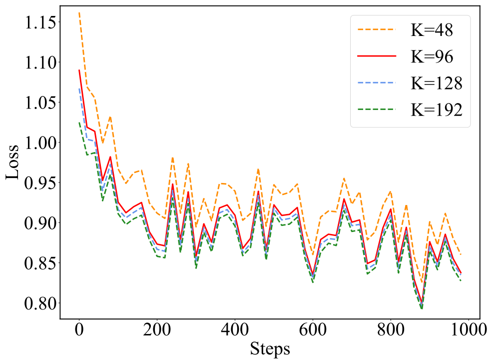


Figure 6: Comparison of fine-tuning loss against different values of K.

We explore ADORE's performance against different K in adaptive token release. In particular, we configure K in the range of $\{48, 96, 128, 192\}$ for fine-tuning the model and top-m as $\{48 \times 2, 96 \times 2, 128 \times 2, 192 \times 2\}$ for a fixed cache size. The inference performance and the corresponding training loss are presented in Table 3 and Figure 6, respectively.

Figure 6 shows that when K values are set to 96, 128, and 192, the differences in training loss are minimal. This indicates that retaining tokens with the highest top-K attention weights is sufficient, and further increasing K does not yield substantial improvements in model performance. From Table 3, it can be observed that there is no significant improvement in the quality of the generated text when m increases from 96×2 to 192×2, which, however, is accompanied by a notable decrease in throughput. Therefore, it is essential to select an appropriate set of K and m, which balances throughput and the quality of generated text.

3.5.2 Influence of KV States Rebuild

We evaluate the impact of different R in KV states rebuild. Specifically, we select R in the range of

m	Throughput	BLEU	ROUGE	BERT-F
48×2	**270.5**	35.5	27.8	62.8
96×2	259.6	36.8	28.8	63.5
128×2	202.6	37.0	29.2	63.7
192×2	167.7	**37.3**	**29.4**	**64.3**

Table 3: Inference performance comparison of maintaining different cache sizes m by adaptive token release. The best results are marked **bold**.

Numbers	Throughput	BLEU	ROUGE	BERT-F
R=0	**278.2**	34.3	26.8	62.3
R=8	259.6	36.8	28.8	63.5
R=16	202.6	37.5	28.9	63.9
R=32	150.8	**38.0**	**29.9**	**64.3**

Table 4: Inference performance comparison of different numbers of rebuilt tokens during inference. The best results are marked **bold**.

$\{0, 8, 16, 32\}$ and summarize the inference performance in Table 4. The results demonstrate that as the R in rebuilt tokens increases, the model's performance first improves. However, the improvement comes at the cost of a reduction in throughput. When the number of rebuilt tokens is further increased from 16 to 32, we can observe an improvement of 1.5% in BLEU, 1.0% in ROUGH, and 0.4% in BERT-F. However, this minor improvement is accompanied by a 34.4% decrease in throughput. This indicates that selecting the appropriate number of rebuilding tokens is crucial for maintaining a trade-off between performance and quality during the inference process.

3.5.3 Effectiveness of Controller Module

Since we use the controller module for advancedly predicting top-K attention weights, next we investigate how it affects overall performance. In particular, we adjust the module with the following variants: **(1)** w/o GRU: directly using the MLP for predicting the keeping/dropping probability of tokens; **(2)** ADORE $_{d'=64}$: set hidden size of the controller to 64; **(3)** ADORE $_{d'=128}$: set hidden size of the controller to 128; We first report accuracy and F1 scores on the dataset that fine-tunes the controller module, as detailed in Section 2.2. Then, we report BLEU, ROUGE, and BERT-F scores on the Ultrachat benchmark, which further illustrate how the performance of the controller module influences the performance of LLMs.

We summarize the results in Table 5. Our observations are as follows: **(1)** The GRU is crucial for the controller module to serve as an effective alternative to full attention; **(2)** an improved controller module results in enhanced performance during the inference process, as it offers a more accurate approximation of sparse attention.

Variants	Controller Acc.	Controller F1	Inference BLEU	Inference ROUGE	Inference BERT-F
w/o GRU	83.4	78.8	36.2	26.4	61.7
ADORE $_{d'=128}$	**87.9**	**82.3**	**37.5**	**28.9**	**63.9**
ADORE $_{d'=64}$	81.5	74.0	33.5	28.5	62.4

Table 5: Performance comparison of different variants controller module and inference. The best results are marked **bold**.

4 Related Work

In this section, we introduce the related work, including sparse attention, efficient LLMs and length extrapolation.

4.1 Sparse Attention

Several works have attempted to integrate sparse attention into transformer-based models. This integration reduces the computational complexity from quadratic to approximately linear in the sequence length, making it possible to process longer sequences. Some studies adopt fixed-pattern sparse strategies (Zaheer et al., 2020; Beltagy et al., 2020), while others focus on sparsification based on the distribution and features of self-attention (Rao et al., 2021; Xiao et al., 2023; Liu et al., 2023b). However, the methods often fail to result in a practical improvement in the inference speed of language models (Ren et al., 2023). This is because the reduction in the number of tokens does not yield significant benefits on CUDA (Bolya et al., 2022). To address this issue, in the LLM inference process, we propose applying dynamic sparse attention to the storage of the key-value (KV) cache, thereby fundamentally enhancing the throughput of the LLM.

4.2 Efficient Inference for LLMs

The efficiency improvement of LLM inference is becoming increasingly attention-grabbing (Huang and Chang, 2023). Recent research has primarily focused on two aspects: systems and algorithms, aiming to enhance LLM inference efficiency. In recent years, numerous systems dedicated to LLM inference have emerged, such as FasterTransformer, Hugging Face Accelerate (Gugger et al., 2022), FlexGen (Sheng et al., 2023), and vLLM (Kwon et al., 2023). These systems often emphasize optimization from hardware accelerators and CUDA kernels. On the other hand, algorithms like Early-Exit (Sun et al., 2021; Rotem et al., 2023) Flashattention-2 (Dao, 2023) and Continuous Batch (Yu et al., 2022) attempt to optimize LLM inference performance by reducing computational costs. In this paper, our proposed method is orthogonal to all mainstream LLM inference systems and most algorithmic optimizations, and our method can be used in parallel with these methods.

4.3 Length Extrapolation for LLM Inference

Length extrapolation aims to enable language models to maintain satisfactory performance when applied to super-long sequences as well. Current mainstream research primarily focuses on finding improved representations for positional encoding. Rotary Position Embeddings (RoPE) (Su et al., 2024) attempt to transform absolute positions into relative position encodings for length expansion. Furthermore, ALiBi (Press et al., 2021) introduces relative positional information by imposing a penalty bias proportional to the distance in relative proximity on the attention matrix. However, current pproaches still struggle to model extremely long texts effectively. Simultaneously, when dealing with long texts, a major limiting factor often lies in GPU memory overflow issues. In this paper, our approach extends the inference length of LLM by setting a fixed attention window size by adaptively releasing tokens, which is designed to maximize the length of inference without compromising performance significantly.

5 Conclusion

We propose an efficient sparse attention for the inference process of LLMs. This is achieved by adaptively releasing the KV state of the tokens with the lowest attention contribution in the cache while simultaneously rebuilding the state of tokens with the highest contribution during the step-by-step decoding of each token. Experimental results show that our approach significantly enhances the throughput of model inference without substantially compromising the quality of the generated text.

6 Limitations

In this paper, the primary limitation lies in the fine-tuning process required to align with our designed inference optimization method. Specifically, during fine-tuning, we still face an $O(n^2)$ time complexity for self-attention, resulting in no speed improvement when learning dynamic sparse attention. Furthermore, our method is not immediately applicable during inference; it requires additional computational overhead for fine-tuning and training the controller to attain enhanced performance during inference.

References

Arkadeep Acharya, Brijraj Singh, and Naoyuki Onoe. 2023. Llm based generation of item-description for recommendation system. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1204–1207.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

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

Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. 2021. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow. If you use this software, please cite it using these metadata.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. 2022. Token merging: Your vit but faster. *arXiv preprint arXiv:2210.09461*.

Alexander Borzunov, Dmitry Baranchuk, Tim Dettmers, Max Ryabinin, Younes Belkada, Artem Chumachenko, Pavel Samygin, and Colin Raffel. 2022. Petals: Collaborative inference and fine-tuning of large models. *arXiv preprint arXiv:2209.01188*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *URL https://openai.com/blog/sparse-transformers*.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Rahul Dey and Fathi M Salem. 2017. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*.

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.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. `https://github.com/huggingface/accelerate`.

Ali Hassani, Steven Walton, Jiachen Li, Shen Li, and Humphrey Shi. 2023. Neighborhood attention transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6185–6194.

Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards reasoning in large language models: A survey. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1049–1065, Toronto, Canada. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Seongsoo Kim, Hayden Wimmer, and Jongyeop Kim. 2022. Analysis of deep learning libraries: Keras, pytorch, and mxnet. In *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 54–62. IEEE.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023a. How long can context length of open-source llms truly promise? In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023b. Camel: Communicative agents for" mind" exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*.

Haoxin Li, Phillip Keung, Daniel Cheng, Jungo Kasai, and Noah A. Smith. 2023c. NarrowBERT: Accelerating masked language model pretraining and inference. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1723–1730, Toronto, Canada. Association for Computational Linguistics.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023d. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6342–6353, Singapore. Association for Computational Linguistics.

Xiaobo Liang, Juntao Li, Lijun Wu, Ziqiang Cao, and Min Zhang. 2023. Dynamic and efficient inference for text generation via BERT family. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2883–2897, Toronto, Canada. Association for Computational Linguistics.

Nora Freya Lindemann. 2023. Sealed knowledges: A critical approach to the usage of llms as search engines. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, pages 985–986.

Songhua Liu, Jingwen Ye, Sucheng Ren, and Xinchao Wang. 2022. Dynast: Dynamic sparse transformer for exemplar-guided image generation. In *European Conference on Computer Vision*, pages 72–90. Springer.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2023a. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023b. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*.

Ofir Press, Noah A Smith, and Mike Lewis. 2021. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. 2021. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *Advances in neural information processing systems*, 34:13937–13949.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Siyu Ren, Qi Jia, and Kenny Zhu. 2023. Context compression for auto-regressive transformers with sentinel tokens. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12860–12867, Singapore. Association for Computational Linguistics.

Daniel Rotem, Michael Hassid, Jonathan Mamou, and Roy Schwartz. 2023. Finding the SWEET spot: Analysis and improvement of adaptive inference in low resource settings. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14836–14851, Toronto, Canada. Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Tianxiang Sun, Yunhua Zhou, Xiangyang Liu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. 2021. Early exiting with ensemble internal classifiers. *arXiv preprint arXiv:2105.13792*.

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

Xinjian Wu, Fanhu Zeng, Xiudong Wang, Yunhe Wang, and Xinghao Chen. 2023. Ppt: Token pruning and pooling for efficient vision transformers. *arXiv preprint arXiv:2310.01812*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. 2023. Predictive pipelined decoding: A compute-latency trade-off for exact llm decoding. *arXiv preprint arXiv:2307.05908*.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538.

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

Dong Zhang, Shimin Li, Xin Zhang, Jun Zhan, Pengyu Wang, Yaqian Zhou, and Xipeng Qiu. 2023. Speechgpt: Empowering large language models with intrinsic cross-modal conversational abilities. *arXiv preprint arXiv:2305.11000*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115.

Lei Zhu, Xinjiang Wang, Zhanghan Ke, Wayne Zhang, and Rynson WH Lau. 2023. Biformer: Vision transformer with bi-level routing attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10323–10333.

A Dataset Statistics

In Table 6, we present the statistical information of the datasets used in our experiments, including dataset partitioning and sequence length statistics.

Dataset	Dataset partitioning			Sequence Length		
	Train	Valid	Test	Average	Median	90 percentile
UltraChat	696600	77400	77400	1476	1411	2265
EverythingLM	972	108	108	1743	1765	2550
Math	45000	5000	5000	510	459	910
StreamEval	2825	353	352	1686	1679	2160
CNN Daily Mail	287113	13368	11490	1132	1060	1825
SAMSum	14700	818	819	3227	3212	3900

Table 6: Performance comparison on StreamChat across different length subsets. The best results are shown in bold.

B Implementation Details

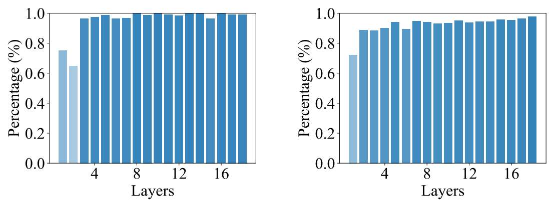
In this section, we illustrate the details of our implementatio, primarily encompassing training data collection, fine-tuning with QLoRA, details of the controller module, inference settings, and hardware settings.

Training data collection As detailed in Section 2.2, it is imperative to gather word embeddings and token indices that contribute to the top-K attention for the current word for training our controller module. Furthermore, during the fine-tuning phase of our model, attention masks are generated utilizing the collected indices to align with the inference process. Specifically, during the fine-tuning process of the full attention (baseline) on the training set, we collect the word embeddings and the most frequently top-K indices of each sample. Given that full attention encompasses the entire sequence, it consistently yields the lowest loss during fine-tuning, thereby ensuring that the attention distribution modeled is reliable and informative for capturing the top-K indices.

Fine-tuning with QLoRA (i) Hyper-parameters: For all methods, we utilize the Adam optimizer with a learning rate of 3e-5, decayed by a rate of 0.98 every 40 steps. Regarding the parameters for Q-LORA, we uniformly set the rank parameter $r = 16$ and the learning rate scaling factor $lora\_alpha = 32$. (ii) Alignment fine-tuning with ADORE: By collecting the top-K indices, we create attention masks for full attention, which block the attention from current token to the low-contribution tokens. This implementation achieves dynamic sparse attention during fine-tuning, resulting in a model aligned with our inference optimization approach.

Details of the controller module (a) Controller network structure: (i) Input layer: A GRU layer with an input size of 4096 and a hidden size of 128; (ii) Position layer: A fully connected layer with an input size of 1, projected to 128; (iii) Interaction layer: A fully connected layer with a hidden size of 128 and a Tanh activation function; (iv) Output layer: Each output, mapped to [0,1] for cross-entropy loss over the sequence length, is obtained through a fully connected layer followed by a sigmoid function. (b) Training Details: We employ the Adam optimizer with a learning rate of 0.005, accompanied by a decay rate of 0.98 every 2000 steps. We split the collected dataset into a training set and a validation set with an 8:2 ratio, and save the model parameters achieving the highest F1 score on the validation set.

Hardware settings We utilized four GeForce RTX 3090 GPUs, with a total runtime exceeding 20 hours.



(a) Proportion of overlapping tokens between the uniform token set and top-$K/2$ tokens sets at each layer.

(b) Proportion of cumulative softmax scores from the uniform token set at each layer.

Figure 7: The effectiveness of the uniform token set at each layer.

C Analysis of Dynamic Sparse Attention

In this section, we first demonstrate the effectiveness of applying a uniform scheduling policy across different layers. Then we showcase the superior performance of dynamic sparse attention in comparison to other methods and delve into the underlying reasons behind its effectiveness.

According to the setup in Section 2.2, we select the top-$K/2$ tokens sets for each layer and consider the top-K tokens that appear most frequently in these sets as the uniform token sets. In Figure 7(a), we observe that the uniform token set covers the majority of the top-K/2 token sets at each layer.

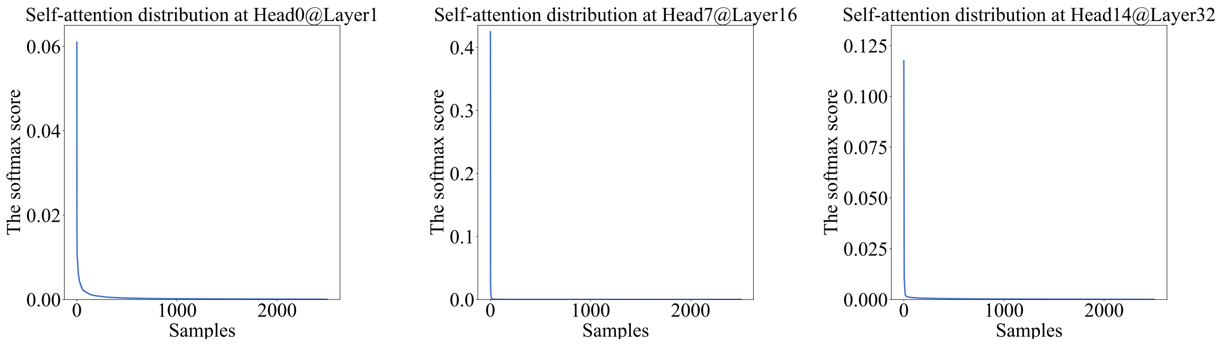


Figure 8: The Softmax scores in the self-attention from a 32-layer Transformer on CNN-DM dataset.

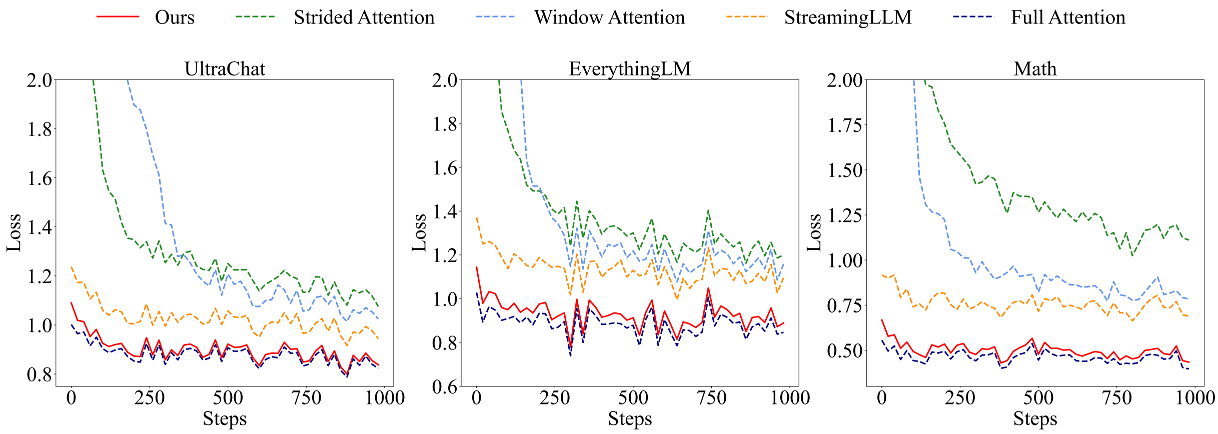


Figure 9: Comparison of loss during fine-tuning across different methods.

Additionally, in Figure 7(b), we illustrate the cumulative softmax attention scores from the uniform token set for the current token across different layers, demonstrating that the uniform token set can effectively replace the contributions of the top-K/2 token sets at each layer.

Figure 9 illustrates the comparison of loss with QLoRA fine-tuning for various methods on UltraChat, EverythingLM, and Math. It is evident that the loss by focusing on the tokens with the top-K highest attention (dynamic sparse attention) maintains consistency with the full attention approach and results in a notable reduction in loss compared to other methods.

The superior performance of dynamic sparse attention can be attributed to the observation that only a small portion of tokens significantly contributes to the attention mechanism during the modeling process for the current token. The Softmax scores in the self-attention curve on the cnn-daily dataset is presented in Figure 8. It can be observed that the blue curve in Figure 8 form a long-tail distribution. This observation provides a reasonable explanation for effective dynamic sparse attention and further demonstrates its good performance in the task.

D Comparison of unidirectional and bidirectional GRU performance

In this section, we compare the performance of unidirectional and bidirectional GRU in terms of top-K prediction accuracy and computation time for 1,000 tokens to illustrate why we choose unidirectional GRU as the primary architecture for the controller module.

Table 7 demonstrates the accuracy and precision on the collected data validation set. We can observe that bidirectional GRU does not significantly improve performance compared to unidirectional GRU. Instead, bidirectional GRU is more computationally expensive in terms of runtime because it requires forward and backward computations at each time step.

Model	Acc.	F1	Time
unidirectional GRU	87.9	82.3	120.5
bidirectional GRU	88.4	83.8	177.3

Table 7: Performance comparison of unidirectional and bidirectional GRU