EdgeInfinite: A Memory-Efficient Infinite-Context Transformer for Edge Devices

Jiyu Chen[*1,2], **Shuang Peng**[*1], **Daxiong Luo**[*1], **Fan Yang**[1], **Renshou Wu**[1], **Fangyuan Li**[1✉], **Xiaoxin Chen**[1]
[1]vivo AI Lab, [2]Zhejiang University
[*]Equal contribution ✉Corresponding author
jiyuchen@zju.edu.cn, {pengshuang,luodaxiong,lifangyuan}@vivo.com

Abstract

Transformer-based large language models (LLMs) encounter challenges in processing long sequences on edge devices due to the quadratic complexity of attention mechanisms and growing memory demands from Key-Value (KV) cache. Existing KV cache optimizations struggle with irreversible token eviction in long-output tasks, while alternative sequence modeling architectures prove costly to adopt within established Transformer infrastructure. We present EdgeInfinite[1], a memory-efficient solution for infinite contexts that integrates compressed memory into Transformer-based LLMs through a trainable memory-gating module. This approach maintains full compatibility with standard Transformer architectures, requiring fine-tuning only a small part of parameters, and enables selective activation of the memory-gating module for long and short context task routing. The experimental result shows that EdgeInfinite achieves comparable performance to baseline Transformer-based LLM on long context benchmarks while optimizing memory consumption and time to first token.

1 Introduction

The Transformer (Vaswani et al., 2017) has become the foundational framework for Large Language Models (LLMs). However, the quadratic time complexity of the classic attention mechanism in Transformer-based model presents significant challenges in processing long sequences. Moreover, the continuous growth of the Key-Value (KV) cache, driven by increasing context lengths, leads to increased memory usage. Whether in terms of time complexity or limited memory, these challenges are particularly pronounced on resource-constrained edge devices such as smartphones.

To address these challenges, two main solutions have been proposed. One approach focuses on the KV cache optimizations (Li et al., 2024b; Xiao et al., 2023; Zhang et al., 2023), primarily by evicting tokens deemed unimportant to reduce attention computation complexity. Though these methods can improve efficiency, they may encounter a potential issue that the evicted tokens will not be used in the future (Tang et al., 2024), especially in real-world scenarios, such as multi-round interactions (Li et al., 2024a; Qin et al., 2024) and long-generation Chain-of-Thought (CoT) reasoning (Wei et al., 2022; Guo et al., 2025).

The second solution explores more efficient sequence modeling methods, such as linear recurrent models (Katharopoulos et al., 2020; Li et al., 2025) and state space models (Gu et al., 2021; Gu and Dao, 2023), to address computational complexity issues. However, most current work remains centered around Transformer-based models. Adopting new structural models would incur substantial costs, hindering their deployment on edge devices.

In this work, we propose **EdgeInfinite**, a novel approach that efficiently handles long sequences on edge devices. By continuing pre-training with existing Transformer-based LLMs, EdgeInfinite maintains compatibility with current Transformer architecture, enabling a more streamlined and resource-efficient approach to model development. We design a trainable memory-gating module that requires fine-tuning only a small subset of parameters. This module can be selectively loaded for long text tasks, while retaining the original parameters of the Transformer model for short text tasks. This flexibility ensures that the base model's parameters do not require additional fine-tuning, allowing for rapid and efficient inference on long text tasks. As a result, our approach is well-suited for deployment on edge devices. During inference, we retain sink tokens and window tokens in KV cache, while the other KV pairs are compressed into the memory block. This approach allows the model to preserve more semantic and positional information during

[1]The code will be released after the official audit.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track), pages 568–575
July 28-30, 2025

inference. Moreover, EdgeInfinite demonstrates the improvement in time to first token (TTFT), a notable advancement among existing methods.

Our contributions can be summarized as follows:

- We propose EdgeInfinite, an edge-side infinite context method that integrates compressed memory with a trainable memory-gating module, while maintaining compatibility with the vanilla Transformer architecture.
- EdgeInfinite maintains the original Transformer-based LLM's performance on short text tasks while supporting high-efficiency inference for long text tasks. This mechanism is highly suitable for model deployment on resource-constrained edge devices.
- We evaluate the performance of EdgeInfinite on long context benchmark. It achieves performance comparable to the baseline Transformer-based models while optimizing memory consumption and TTFT.

2 Related work

The quadratic time complexity of the attention mechanism and the growing memory use of the KV cache in classic Transformer-based LLMs pose challenges for processing long sequences on resource-constrained edge devices. This section highlights recent work to address these issues.

Innovative Sequence Models Mamba (Gu and Dao, 2023) and Mamba-2 (Dao and Gu, 2024) represent the significant milestone in the development of State Space Model (SSM) (Gu et al., 2021), demonstrating outstanding performance in natural language processing and other tasks. The RWKV (Peng et al., 2023, 2024) combines the advantages of RNN and Transformer, introducing innovations such as token shift and optimized time-mixing to achieve linear complexity in inference. Titans (Behrouz et al., 2024) combine attention as short-term memory with a neural long-term memory module. Infini-Transformer (Munkhdalai et al., 2024) segments long sequences into multiple blocks, incorporates a compressive memory into the vanilla attention mechanism and builds in both masked local attention and long-term linear attention mechanisms in a single Transformer block.

KV cache Optimizations KV Cache Optimizations primarily aim to reduce overall computational requirements by identifying and discarding unimportant tokens. StreamingLLM (Xiao et al., 2023) is a method based on sliding window attention. By retaining both the most recent and sink tokens, it helps maintain the model's performance while efficiently managing memory usage. H2O (Zhang et al., 2023) employs attention scores to identify and retain significant tokens while simultaneously preserving the most recent tokens. SnapKV (Li et al., 2024b) identifies critical attention features based on observation windows and correspondingly compresses the KV cache. PyramidKV (Cai et al., 2024) reduces the KV cache budget for later layers by analyzing the attention features across different layers. SCOPE (Wu et al., 2024) innovatively refines the KV cache budget problem by considering it separately in the prefill and decode stages.

3 EdgeInfinite

3.1 Architecture

As shown in Figure 1, the architecture of EdgeInfinite includes three core components: (1) Segmented attention with Rotary Position Embedding (ROPE) for local context modeling, (2) The memory mechanism for compressing and decompressing historical context, and (3) The adaptive memory-gating module that balances local and memory-based attention.

3.1.1 Segmented Attention with ROPE

Given an input sequence $X = [x_1, \ldots, x_L]^T \in \mathbb{R}^{L \times d}$, it is divided into segments of size L_{seg}, resulting in N segments of length L_{seg} and a residual segment of length L_{res}. Their relationship can be expressed as:

$$L = N \cdot L_{seg} + L_{res} \tag{1}$$

The full segment $X_{seg} \in \mathbb{R}^{L_{seg} \times d}$ or the residual segment $X_{res} \in \mathbb{R}^{L_{res} \times d}$ can be collectively represented as $X_{s/r} \in \mathbb{R}^{L_{s/r} \times d}$, where s/r indicates either a full or residual segment. We compute the attention query Q, key K, and value V states:

$$Q = X_{s/r}W^Q, K = X_{s/r}W^K, V = X_{s/r}W^V \tag{2}$$

where W^K, W^V, and W^Q are the trainable projection matrices. $Q = [q_1, q_2, \ldots, q_{L_{s/r}}]$ and $K = [k_1, k_2, \ldots, k_{L_{s/r}}]$ denote the query and key states in the segment $X_{s/r}$, where q_i and k_i represent the query and key states corresponding to the i-th token.

Next, the ROPE model (Su et al., 2024) is integrated to incorporate positional information into the attention computation:

$$q_m^r = \mathcal{R}_m q_m, k_n^r = \mathcal{R}_n k_n \tag{3}$$

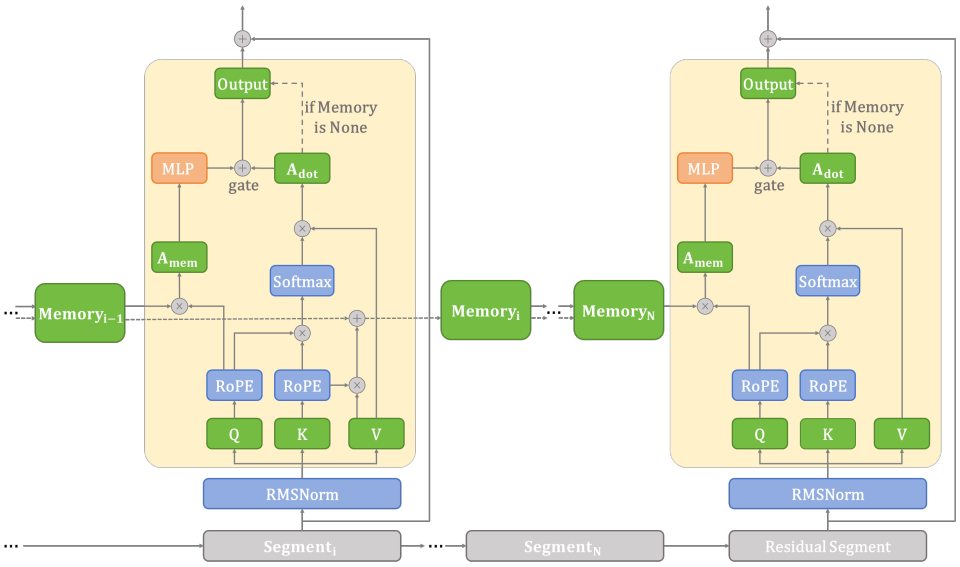

Figure 1: The overall framework of EdgeInfinite: illustrating the computation process of the attention layer in Transformer-based LLMs, with LLaMA Attention (Touvron et al., 2023; Grattafiori et al., 2024) as an example.

where R_m and R_n are the rotary matrices situated at positions m and n. q_m^r and k_n^r represent the query and key states after the rotary transformation. After applying the rotary transformation, the modified query and key states are denoted as Q^r and K^r.

Subsequently, the attention computation for each segment is performed in a manner similar to the vanilla Transformer architecture (Vaswani et al., 2017):

$$A_{\text{dot}} = \text{softmax}(\frac{Q^r(K^r)^T}{\sqrt{d}})V \tag{4}$$

This computation enables the model to capture dependencies between tokens within each segment while incorporating positional information through the ROPE model.

3.1.2 Memory Compression-Decompression

Inspired by the Infini-Transformer (Munkhdalai et al., 2024) and linearized attention (Katharopoulos et al., 2020), we introduce memory compression and memory decompression. For all segments except the residual segment, memory compression is performed. For the i-th segment, the memory M_i and the normalization term z_i are calculated as follows:

$$M_i = M_{i-1} + \sigma(K^r)^T V \tag{5}$$

$$z_i = z_{i-1} + \sum_{j=1}^{L_{s/r}} \sigma(k_j^r) \tag{6}$$

where σ denotes a nonlinear activation function. $M_i \in \mathbb{R}^{d\times d}$ and $z_i \in \mathbb{R}^{d\times 1}$ are both initialized as zero matrices for the first segment ($i = 1$). Here, the memory M_i stores the associations between the keys and values of previous segments. The nonlinear activation function and normalization are primarily used to ensure the stability of model training.

For all segments, the memory decompression is executed as follows:

$$A_{\text{mem}} = \frac{\sigma(Q^r)M_{i-1}}{\sigma(Q^r)z_{i-1}} \tag{7}$$

where $A_{\text{mem}} \in \mathbb{R}^{L_{r/s}\times d}$ represents the attention calculated by the memory and query state of the current segment. Since the memory encodes the associations of key-value pairs from previous segments, decompression allows us to compute the attention between the current query state and the past key-value states. This process enables blockwise computation to approximate the attention calculation of the original long sequence.

3.1.3 Memory-Gating Module

In contrast to the Infini-Transformer, which requires training the entire model, our approach requires fine-tuning only the memory-gating module. This module can integrate memory-based attention with local segment-based attention, enhancing the model's ability to handle long-range dependencies. Additionally, our method supports switching to the original model for inference on short context tasks.

The memory-gating module is a trainable component that consists of a Multi-Layer Perceptron (MLP) and a gating vector. Specifically, the memory attention A_{mem} is first transformed through the MLP as follows:

$$\tilde{A}_{\text{mem}} = W_2 \cdot \text{ReLU}(W_1 A_{\text{mem}} + b_1) + b_2 \quad (8)$$

Here, W_1 and W_2 are trainable weight matrices, while b_1 and b_2 are bias vectors. The ReLU activation function introduces non-linearity, enabling the MLP to refine the memory-based attention and capture complex interactions between the current segment and accumulated memory.

The transformed memory attention $\tilde{A}_{\text{mem}}$ is then combined with the local segment-based attention A_{dot} through a gating mechanism. The combined attention A_{com} is computed as:

$$\begin{aligned} A_{\text{com}} = &\ \text{sigmoid}(g) \odot \tilde{A}_{\text{mem}} \\ &+ (1 - \text{sigmoid}(g)) \odot A_{\text{dot}} \end{aligned} \quad (9)$$

where g is a trainable gating vector. The sigmoid function applied to g produces a gating factor that adaptively controls the contribution of $\tilde{A}_{\text{mem}}$ and A_{dot} to the combined attention. This adaptive weighting mechanism ensures that the model can dynamically balance the importance of previous context (encoded in $\tilde{A}_{\text{mem}}$) and current context (encoded in A_{dot}) based on the specific features of the long sequence.

The memory-gating module is integrated as a bypass in the attention pipeline. If the sequence length is insufficient to be divided into segments, the memory is None and the memory mechanism is bypassed, reverting to standard Multi-Head Attention. The final attention output O is given by:

$$\begin{cases} O = [A^1_{\text{com}}; \dots A^H_{\text{com}}]W_o & \text{if Memory} \neq \text{None} \\ O = [A^1_{\text{dot}}; \dots A^H_{\text{dot}}]W_o & \text{if Memory} = \text{None} \end{cases} \quad (10)$$

where A^h_{com} and A^h_{dot} represent the combined attention and the local segment-based attention for the h-th head. This design ensures consistency with the base model for short context tasks, avoiding catastrophic forgetting.

3.2 Inference Strategy

The inference strategy of EdgeInfinite is formalized in Algorithm 1 and visualized in Figure 2. It is characterized by two main components: (1) Selective token preservation to ensure high-quality inference, and (2) Adaptive long-short text routing for handling of diverse input lengths.

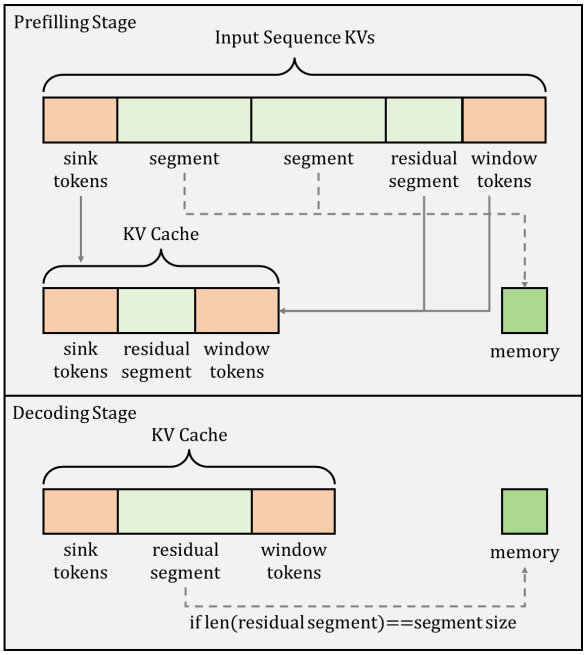

Figure 2: The inference strategy of EdgeInfinite.

3.2.1 Selective Token Preservation

EdgeInfinite significantly compresses the key states and value states associated with multiple tokens, similar to KV cache optimization methods that discard several tokens to reduce computational overhead. However, this approach may potentially degrade overall inference performance.

To address this issue, EdgeInfinite preserves two types of important tokens in the KV cache during the inference process: **sink tokens** and **window tokens**. Sink tokens represent the initial tokens of the sequence, while window tokens correspond to the most recent tokens. These tokens are crucial for preserving semantic and positional information (Xiao et al., 2023), and they are retained uncompressed to ensure high-quality inference outputs.

3.2.2 Long-Short Text Inference Routing

EdgeInfinite's inference strategy adapts dynamically to handle both long and short text inputs efficiently. The entire inference process can be divided into prefilling stage and decoding stage:

Prefilling Stage For long input sequences ($L \geq L_{\text{sink}} + L_{\text{window}} + L_{\text{seg}}$), the sequence excluding the sink tokens and window tokens is divided into N chunks, each of length L_{seg}. Each chunk is compressed into memory, with sink tokens concatenated in front. The remaining parts, including the residual segment, are stored as KV cache. For short input sequences ($L < L_{\text{sink}} + L_{\text{window}} + L_{\text{seg}}$), the model retains the full KV cache, similar to the

Algorithm 1 EdgeInfinite Inference Strategy.

Input: Input sequence $X_{\text{in}} = [x_1, \ldots, x_L]^T$, memory M, normalization term z, KV cache C
Output: Output sequence $X_{\text{out}} = [x_1, \ldots, x_{L_{\max}}]^T$
// Prefilling stage:
Initialize memory M, normalization term z, and KV cache C
if $L \geq L_{\text{sink}} + L_{\text{window}} + L_{\text{seg}}$ **then**
  $C = \text{get\_kv\_cache}(X_{\text{in}}[: L_{\text{sink}}], C)$
  $N = \lfloor (L - L_{\text{sink}} - L_{\text{window}})/L_{\text{seg}} \rfloor$
  for $i = 0$ **to** $N - 1$ **do**
    $X_{\text{segment}} = X_{\text{in}}[L_{\text{sink}} + i \cdot L_{\text{seg}} : L_{\text{sink}} + (i+1) \cdot L_{\text{seg}}]$
    $M, z = \text{get\_memory}(X_{\text{segment}}, C, M, z)$
  end for
  $X_{\text{remaining}} = X_{\text{in}}[L_{\text{sink}} + N \cdot L_{\text{seg}} :]$
  $O, C = \text{get\_model\_output}(X_{\text{remaining}}, C, M, z)$
else
  $O, C = \text{get\_model\_output}(X_{\text{in}}, C, M, z)$
end if
$x_{\text{new}} = \text{get\_model\_decode}(O)$
$X_{\text{out}} = [X_{\text{in}}; x_{\text{new}}]$
// Decoding stage:
while $\text{len}(X_{\text{out}}) < L_{\max}$ **do**
  $L_{\text{res}} = \text{len}(X_{\text{out}}) - L_{\text{sink}} - L_{\text{window}}$
  if $L_{\text{res}} == L_{\text{seg}}$ **then**
    $X_{\text{segment}} = X_{\text{out}}[-L_{\text{seg}} - L_{\text{window}} : -L_{\text{window}}]$
    $C = C[: L_{\text{sink}}]$
    $M, z = \text{get\_memory}(X_{\text{segment}}, C, M, z)$
    $O, C = \text{get\_model\_output}(X_{\text{out}}[-L_{\text{window}} :], C, M, z)$
  else
    $O, C = \text{get\_model\_output}(X_{\text{out}}[-1 :], C, M, z)$
  end if
  $x_{\text{new}} = \text{get\_model\_decode}(O)$
  $X_{\text{out}} = [X_{\text{out}}; x_{\text{new}}]$
end while

original attention. Here, L_{sink} and L_{window} are the lengths of retained sink tokens and window tokens.

Decoding Stage The model iteratively generates new tokens until the length of the output sequence reaches L_{max}. If the length of the residual sequence equals L_{seg}, the memory is updated by compressing the corresponding segment, with the sink tokens concatenated in front. The output is then generated based on the updated memory, the KV cache of sink tokens, and the KV states of window tokens. Otherwise, the model directly generates the next token using the current KV cache and memory.

4 Experiments

4.1 Experimental Setups

Model and Data In our experiments, we evaluate EdgeInfinite using BlueLM-3B (Lu et al., 2024) as the backbone, a Transformer-based LLM suitable for edge deployment. The training dataset includes approximately 100,000 samples, covering diverse tasks such as text summarization and generation.

Hyperparameters The model is trained for 2 epochs with a learning rate set to 0.005. Only the memory-gating module (0.15% of total weights) is trained. We configure other hyperparameters as follows: L_{seg} is set to 2048, L_{sink} to 300, and L_{window} to 200. For sequences of varying lengths, the total size of the retained KV cache averages approximately 1524 tokens, which includes 300 sink tokens, 200 window tokens, and an average residual segment length of 1024 tokens.

Benchmark We evaluate EdgeInfinite using LongBench (Bai et al., 2023), a multi-task long-context benchmark for assessing long-context comprehension abilities across diverse datasets.

Baseline We compare EdgeInfinite with three baseline KV cache optimization methods, including SnapKV (Li et al., 2024b), PyramidKV (Cai et al., 2024), and StreamingLLM (Xiao et al., 2023), as well as the original model with full KV cache. The cache sizes for these three baselines are set to 2048, slightly larger than the setting of EdgeInfinite.

4.2 Results

The performance comparison between baseline and our method is shown as Table 1. We report the average performance for each category , as well as the overall average performance across all 21 tasks.

Overall, EdgeInfinite demonstrates competitive performance advantages compared to other baselines and even exceeds the performance of FullKV. In specific tasks, EdgeInfinite demonstrates relatively better performance in summarization and code completion, and achieves notable superior results in multi-document QA and few-shot learning.

It can be revealed that KV cache optimization methods generally perform similarly to or slightly better than FullKV. However, EdgeInfinite significantly outperforms FullKV in certain tasks, such as HotpotQA and TriviaQA. The performance enhancement is attributed to its strategy of segmenting long context sequences into multiple shorter sequences, reducing performance degradation from processing excessively long sequences. Meanwhile, EdgeInfinite shows relatively weaker performance in single-document QA than in multi-document QA. This is because single-document QA requires precise answers, while multi-document QA focuses on summarizing content. The memory compression in EdgeInfinite leads to precision loss in KV states, making it better suited for generating summary answers rather than precise retrieval.

4.3 Ablation Study

To evaluate the impact of retaining specific KV cache during the inference process of EdgeInfinite,

	Single-Document QA					Multi-Document QA					Summarization				
	NrtvQA	Qasper	MF-en	MF-zh	Avg	HotpotQA	2WikiMQA	MuSiQue	DuReader	Avg	GovReport	QMSum	MultiNews	VCSUM	Avg
FullKV	5.94	31.50	34.89	47.88	30.05	21.93	26.15	2.58	24.91	18.89	12.82	7.04	10.94	18.34	12.29
SnapKV	5.53	29.80	35.04	**48.97**	29.84	22.51	26.04	2.14	22.77	18.37	11.09	6.68	**11.08**	17.74	11.65
PyramidKV	5.01	**30.06**	**35.50**	48.82	**29.85**	22.25	25.76	2.22	22.95	18.30	11.27	6.53	10.93	17.60	11.58
StreamingLLM	3.70	25.54	29.45	43.15	25.46	16.63	19.13	2.25	23.61	15.41	10.84	5.27	10.50	17.39	11.00
EdgeInfinite	**14.16**	18.68	25.58	35.56	23.50	**31.67**	**26.08**	**12.06**	**26.87**	**24.17**	**11.28**	**8.18**	10.76	**18.18**	**12.10**

(a) Results on single-document QA, multi-document QA, and summarization tasks.

	Few-shot Learning					Synthetic				Code			Overall
	TREC	TriviaQA	SAMSum	LSHT	Avg	PCount	PRe-en	PRe-zh	Avg	LCC	RB-P	Avg	Avg
FullKV	63.00	51.98	24.50	18.00	39.37	2.50	4.50	28.00	11.67	42.96	27.81	35.39	24.20
SnapKV	60.00	51.98	24.32	17.75	38.51	1.79	5.50	**30.00**	**12.43**	43.72	27.07	35.40	23.88
PyramidKV	**61.00**	51.46	24.07	18.00	38.63	2.17	5.31	28.50	11.99	**43.86**	26.74	35.30	23.81
StreamingLLM	**61.00**	38.20	10.92	14.17	31.07	2.60	4.29	7.50	4.80	33.49	22.66	28.08	19.16
EdgeInfinite	55.00	**79.03**	**33.27**	**24.25**	**47.89**	**3.50**	**6.00**	24.00	11.17	42.66	**33.09**	**37.88**	**25.71**

(b) Results on few-shot learning, synthetic, code tasks, and overall LongBench task average results.

Table 1: Performance comparison of EdgeInfinite (Ours) with SnapKV, PyramidKV, StreamingLLM and FullKV on LongBench.

	SQA	MQA	Sum	FS	Syn	Code	Avg
EdgeInfinite	**23.50**	24.17	**12.10**	**47.89**	**11.17**	37.88	**25.71**
wo window tokens	23.28	**24.36**	11.74	46.44	10.00	37.74	25.18
wo sink tokens	20.43	19.12	11.60	43.78	6.12	**44.28**	23.17
wo sink & window	19.06	18.56	11.19	40.10	5.92	43.19	21.90

Table 2: Ablation experiment results on LongBench (SQA = Single-Document QA, MQA = Multi-Document QA, Sum = Summarization, FS = Few-shot Learning, Syn = Synthetic).

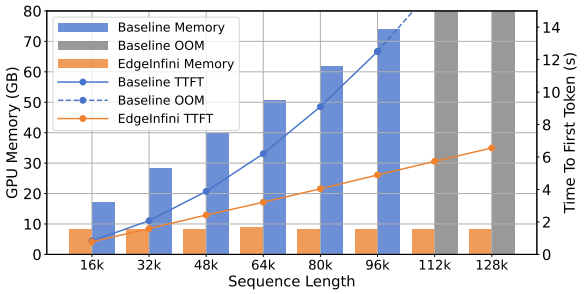

Figure 3: Efficiency of EdgeInfinite. We demonstrate GPU memory consumption and TTFT for varying input sequence lengths.

we conduct ablation studies to assess the effects of sink tokens and window tokens on inference performance. These ablation experiments are also performed on LongBench. Table 2 presents the average scores for different task categories and the overall average score under three conditions: removing sink tokens, removing window tokens, and removing both sink and window tokens.

Removing sink tokens significantly impacts the results of most tasks, as the initial tokens often contain important positional and semantic information for many tasks. Additionally, removing window tokens also affects overall performance. Retaining a fixed number of window tokens avoids the issue of L_{res} being too short, which would result in too few tokens retained as KV cache at the end of the sequence during memory compression. This mechanism effectively maintains semantic continuity during inference.

4.4 Efficiency

We compare TTFT and memory usage between EdgeInfinite and the original BlueLM-3B model, as shown in Figure 3. The results demonstrate that EdgeInfinite exhibits significant advantages in handling long sequences, with resource consumption not increasing rapidly with sequence length. This is attributed to our method's ability to process long sequences in chunks within the segment size, thereby substantially reducing the computational resource requirements.

5 Conclusion

In this study, we propose EdgeInfinite, an efficient method for long context tasks on edge devices. By integrating compressed memory into the Transformer-based LLMs with a trainable memory-gating module, we enable efficient inference on infinite context while maintaining compatibility with the vanilla Transformer architecture. Additionally, we design an effective strategy to retain important tokens during inference for long context tasks to enhance the inference performance, and switch to the original backbone model for short context tasks. Our evaluation on long context benchmarks reveals that EdgeInfinite achieves performance comparable to baseline methods. In summary, EdgeInfinite offers an efficient solution for long context tasks on resource-constrained edge devices.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. 2024. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Tri Dao and Albert Gu. 2024. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Albert Gu, Karan Goel, and Christopher Ré. 2021. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*.

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

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR.

Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. 2025. Minimax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, et al. 2024a. Scbench: A kv cache-centric analysis of long-context methods. *arXiv preprint arXiv:2412.10319*.

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

Xudong Lu, Yinghao Chen, Cheng Chen, Hui Tan, Boheng Chen, Yina Xie, Rui Hu, Guanxin Tan, Renshou Wu, Yan Hu, et al. 2024. Bluelm-v-3b: Algorithm and system co-design for multimodal large language models on mobile devices. *arXiv preprint arXiv:2411.10640*.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. 2024. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 101.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. 2023. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Teddy Ferdinan, Haowen Hou, Przemysław Kazienko, et al. 2024. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. *arXiv preprint arXiv:2404.05892*, 3.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu Mooncake. 2024. Kimi's kvcache-centric architecture for llm serving. *arXiv preprint arXiv:2407.00079*.

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

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. 2024. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*.

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*.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. 2024. Scope: Optimizing key-value cache compression in long-context generation. *arXiv preprint arXiv:2412.13649*.

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

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