InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory

Anonymous Authors[1]

Abstract

Large language models (LLMs) have emerged as a cornerstone in real-world applications with lengthy streaming inputs (e.g., LLM-driven agents). However, existing LLMs, pre-trained on sequences with a restricted maximum length, cannot process longer sequences due to the out-of-domain and distraction issues. Common solutions often involve continual pre-training on longer sequences, which will introduce expensive computational overhead and uncontrollable change in model capabilities. In this paper, we unveil the intrinsic capacity of LLMs for understanding extremely long sequences without any fine-tuning. To this end, we introduce a training-free memory-based method, InfLLM. Specifically, InfLLM stores distant contexts into additional memory units and employs an efficient mechanism to lookup token-relevant units for attention computation. Thereby, InfLLM allows LLMs to efficiently process long sequences with a limited context window and well capture long-distance dependencies. Without any training, InfLLM enables LLMs that are pre-trained on sequences consisting of a few thousand tokens to achieve comparable performance with competitive baselines that continually train these LLMs on long sequences. Even when the sequence length is scaled to $1,024$K, InfLLM still effectively captures long-distance dependencies. Our code will be released to advance the processing of extremely long sequences.

1. Introduction

Recently, large language models (LLMs) have achieved profound accomplishments in various tasks (Brown et al., 2020; Bommasani et al., 2021; Han et al., 2021; Touvron et al., 2023; Meta, 2024). Their ability to follow complex instructions shed light on the realization of artificial general intelligence (OpenAI, 2023; Ouyang et al., 2022). With the blooming of LLM-driven applications, such as agent construction (Park et al., 2023; Qian et al., 2023; Wang et al., 2024a) and embodied robotics (Driess et al., 2023; Liang et al., 2023), enhancing the capability of LLMs to process streaming long sequences become increasingly crucial. For instance, LLM-driven agents are required to process information continuously received from external environments based on all their historical memories, necessitating a robust capability for handling long streaming sequences.

Due to limitations caused by unseen lengthy inputs (Han et al., 2023) and distracting noisy contexts (Liu et al., 2023; Tworkowski et al., 2023), most LLMs, pre-trained on sequences consisting of only a few thousand tokens, cannot process longer sequences (Press et al., 2022; Zhao et al., 2023). Common solutions usually involve continually training LLMs on longer sequences but further result in substantial costs and require large-scale high-quality long-sequence datasets (Xiong et al., 2023; Li et al., 2023). And the continual training process on longer sequences may weaken the performance of LLMs on short contexts (Ding et al., 2024). In view of this, improving the length generalizability of LLMs without further training receives extensive attention, trying to make LLMs trained on short sequences directly applicable to long sequences.

In this paper, we propose a training-free memory-based approach, named InfLLM, for streamingly processing extremely long sequences with limited computational costs. Specifically, InfLLM incorporate the sliding window attention (Xiao et al., 2023; Han et al., 2023) with an efficient context memory, where each token only attends to local contexts and relevant contexts from the memory. Considering the sparsity of attention score matrices, processing each token typically requires only a small portion of its contexts (Zhang et al., 2023b), and the remaining irrelevant contexts act as noise, leading to attention distraction issues (Tworkowski et al., 2023). We thus construct an external memory containing distant context information. Only relevant information within the memory is selected for each computation step, and other irrelevant noises are ignored.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Under review by the Workshop on Long-Context Foundation Models (LCFM) at ICML 2024. Do not distribute.

Owing to this, LLMs can understand whole long sequences using a finite-size window and avoid noisy contexts.

The vast amount of noisy context tokens in long sequences poses significant challenges to effective and efficient memory lookup. To address these challenges, we design a block-level context memory mechanism. Specifically, InfLLM organizes past key-value vectors into blocks, each containing a continuous token sequence. Within each block, the semantically most significant tokens that receive the highest attention scores are selected as the unit representation for subsequent relevance computation in memory lookup. This design offers two primary benefits: (1) Effective Lookup: The coherent semantics of each block can more effectively fulfill the requirements for relevant information retrieval compared to single tokens. The selection of unit representations minimizes the interference of unimportant tokens in relevance computation, enhancing the overall hit rate of memory lookup. (2) Efficient Lookup: The block-level memory unit eliminates the need for per-token relevance computation, significantly reducing computational costs. Moreover, block-level units ensure contiguous memory access, thus minimizing memory loading costs and enhancing computational efficiency. Furthermore, considering the infrequent usage of most units, InfLLM offloads all units on CPU memory and dynamically retains the frequently used units on GPU memory, significantly reducing GPU memory usage. **Notably, the block-level memory mechanism in InfLLM does not involve any additional training, and can be directly applied to any LLMs.**

We use a widely-used benchmark, ∞-Bench (Zhang et al., 2023a) for evaluation. Especially, the average sequence length in ∞-Bench exceeds 100K tokens, which is challenging for most existing LLMs. The results show the effectiveness of InfLLM. Moreover, we examine InfLLM on the sequences containing $1,024$K tokens, and InfLLM can still effectively capture long-distance dependencies.

2. Methodology

As shown in Figure 1, InfLLM builds a training-free context memory to efficiently provide highly-relevant contexts for each token, endowing the sliding window attention mechanism with the ability to capture long-distance dependencies.

2.1. Overall Framework

The main restrictions for improving the length generalizability of LLMs come from the out-of-domain and distraction issues caused by the lengthy and noisy contexts. To address these, following previous works (Xiao et al., 2023; Han et al., 2023), we adopt the sliding window attention mechanism, which only considers local tokens for each step. Additionally, we construct an extra context memory module to provide relevant context information to capture long-distance dependencies.

Specifically, we denote the long input sequence as $s = \{t_i\}_{i=1}^{l}$. Due to the limited GPU memory, instead of encoding the whole s at once, we encode the input sequence s chunk-by-chunk and generate the output token-by-token. For each computation step, the inputs consist of past key-value vectors $\mathbf{P} = \{(\mathbf{k}_j, \mathbf{v}_j)\}_{j=1}^{l_P}$ and current tokens $\mathbf{X} = \{\mathbf{t}_{i+l_P}\}_{i=1}^{l_X}$. For encoding steps, l_X equals the chunk size, and for decoding steps, l_X equals one.

According to the distances from current tokens, we can divide $\mathbf{P}$ into three groups: initial tokens, $\mathbf{I} = \mathbf{P}_{[1:l_I]}$, evicted tokens, $\mathbf{E} = \mathbf{P}_{[l_I+1:l_P-l_L]}$, and local tokens, $\mathbf{L} = \mathbf{P}_{[l_P-l_L+1:l_P]}$, arranged from the furthest to the nearest relative to the current tokens. Here, l_P, l_I, l_L refer to the length of past key-value vectors, initial tokens, and the local window size. All evicted tokens, $\mathbf{E}$, are stored in the context memory, consisting of multiple memory units. For each step, InfLLM concatenates the initial tokens, relevant memories units from context memory, and local tokens to form the current key-value cache, $\mathbf{C} = \text{Concat}(\mathbf{I}, f(\mathbf{X}, \mathbf{E}), \mathbf{L})$. $f(\cdot)$ refers to the lookup operation of context memory. The attention output is calculated as:

$$\mathbf{O} = \text{Attn}\left[\mathbf{QX}, \text{Concat}(\mathbf{C}_k, \mathbf{KX}), \text{Concat}(\mathbf{C}_v, \mathbf{VX})\right].$$

Here, $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are parameters in attention layers, $\mathbf{C}_k$ and $\mathbf{C}_v$ refer to the key and value vectors in $\mathbf{C}$.

2.2. Context Memory

Previous findings indicate that the attention score matrices of LLMs are sparse (Zhang et al., 2023b). Inspired by this, we design a context memory to efficiently look up relevant contexts from large-scale evicted tokens and ignore irrelevant ones to save computational costs. The most intuitive way is to construct a memory consisting of token-level memory units for every past key-value vectors, and every attention head separately, which would result in massive memory units, unacceptable computation, and non-contiguous memory access costs. Thus, considering the local semantic coherence of long sequences, we split the past key-value vectors into blocks, each serving as a memory unit, and conduct memory lookup at the block level to reduce the costs while preserving the performance.

Block-Level Memory Units. Block-level memory units can save computation costs compared to token-level ones. It also poses new challenges for unit representations, which are supposed to contain the semantics of the entire unit for effective relevance score computation and be memory-efficient for context length scalability. Traditional methods usually involve training an additional encoder to project a given unit into a low-dimension vector. Inspired by the token redundancy in hidden states (Goyal et al., 2020; Dai et al., 2020),

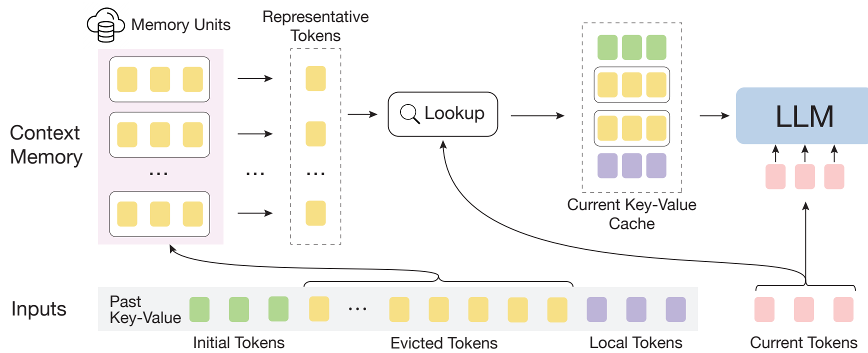

Figure 1: The illustration of InfLLM. Here, the current tokens refer to tokens that need to be encoded in the current computation step. For each step, the context window consists of the initial tokens, relevant memory units, and local tokens.

we select several **representative tokens** from the entail blocks as the unit representation. For the m-th token, we define the representative score as: $r_m = \frac{1}{l_L}\sum_{j=1}^{l_L} \mathbf{q}_{m+j} \cdot \mathbf{k}_m$, where $\mathbf{q}_{m+j}$ is the query vector for $(m+j)$-th token and $\mathbf{k}_m$ is the key vector m-th token. Intuitively, r_m represents the significance of the m-th token in its corresponding local window, indicating the extent of its influence on other tokens within the local window. The computation of representative scores requires no additional parameters.

Formally, given the evicted tokens, $\mathbf{E}$, we split it into several memory units, each containing l_{bs} tokens. For each unit, the r_k tokens with the highest representative scores are selected as representative tokens. Generally, r_k is a small positive integer. Let us denote a memory unit as $\mathbf{B} = \{(\mathbf{k}_j^B, \mathbf{v}_j^B)\}_{j=1}^{l_{bs}}$, and the representative tokens of this unit as $R(\mathbf{B}) = \{(\mathbf{k}_{b_j}^B, \mathbf{v}_{b_j}^B)\}_{j=1}^{r_k}$.

For the **memory lookup** phrase, only k_m units with the highest relevance scores are loaded for the current attention computation. We calculate the relevance score between $\mathbf{B}$ and current tokens $\mathbf{X}$ as: $\text{sim}(\mathbf{X}, \mathbf{B}) = \sum_{i=1}^{l_X}\sum_{j=1}^{r_k} \mathbf{q}_{i+l_P} \cdot \mathbf{k}_{b_j}^B$. Notably, the representative tokens selection is a training-free method to obtain the unit representations. Here, we can also train an additional encoder to generate more expressive unit representations, which we leave for future work.

Positional Encoding. Existing LLM training usually employs a finite number of positional encodings, which encounter out-of-domain distribution challenges when directly applied to longer sequence processing (Han et al., 2023). We assign all tokens beyond the local window size with the same positional encodings, and the distance between tokens in context memory units and current tokens is set as l_L.

Cache Management. To enable LLMs to process extremely long sequence streams while capturing the semantic relevance contained in the long contexts, we need to retain all memory units and look up them at each computation step. Considering the infrequent usage of most units, we employ an offloading mechanism, storing most memory units in CPU memory and only preserving the representative tokens and memory units needed in current steps in GPU memory. Additionally, given the semantic coherence of long sequences, where adjacent tokens often require similar memory units, we allocate a cache space in GPU memory, managed using a least recently used strategy. This approach allows for efficient encoding of extremely long sequences using limited GPU memory. From the observation, our offloading mechanism enables InfLLM to process sequences consisting of 100K tokens with only 26G VRAM. Besides, the miss rate of our GPU cache is quite low, which means the offloading mechanism does not introduce significant time overhead in memory loading while saving GPU memory usage. The details can be found in the Appendix.

3. Experiments

3.1. Settings

Datasets. We adopt representative tasks in a widely-used long document benchmark, ∞-Bench (Zhang et al., 2023a) for evaluation. The average length for ∞-Bench is 145.1K. The 95% quantile for sequence lengths is 214K, which is far beyond the maximum length of the base models.

Baseline Models. To verify the effectiveness of our proposed method, we compare InfLLM with the following competitive baseline models: (1) **Original** models; (2) Position downscaling and resuing: NTK-aware scaled RoPE (**NTK**) (LocalLLaMA, 2023) and **Self-Extend** (Jin et al., 2024); (3) Sliding window: LM-Infinite (**Infinite**) (Han et al., 2023) and StreamingLLM (**Stream**) (Xiao et al., 2023); (5) Key-value eviction: **H2O** (Zhang et al., 2023b). The detailed description about the baseline models can be found in the Appendix.

3.2. Main Results

The results for Mistral-based models and Llama-3-based models are reported in Table 1. From the results, we can ob-

Table 1: The results of InfLLM and baseline models on ∞-Bench. The context window size for sliding window models refers to the local window size, and for InfLLM refers to "local window size + selected memory size".

	Window	Streaming	R.PK	R.Num	R.KV	Choice	QA	Sum	Math.F	Avg.
Mistral-based Models (7B)										
Mistral	32K	✗	28.8	28.8	14.8	44.5	12.9	25.9	20.6	25.2
NTK	128K	✗	100.0	86.8	19.2	40.2	16.9	20.3	26.9	44.3
SelfExtend	128K	✗	100.0	100.0	15.6	42.8	17.3	18.8	19.1	44.8
Infinite	32K	✓	28.8	28.8	0.4	42.8	11.4	22.5	16.3	21.6
Streaming	32K	✓	28.8	28.5	0.2	42.4	11.5	22.1	16.9	21.5
H2O	32K	✓	8.6	4.8	2.6	48.0	15.6	24.4	26.9	18.7
InfLLM	16K	✓	100.0	96.1	96.8	43.7	15.7	25.8	25.7	57.7
Llama-3-based Models (8B)										
Llama-3	8K	✗	8.5	7.8	6.2	44.1	15.5	24.7	21.7	18.4
NTK	128K	✗	0.0	0.0	0.0	0.0	0.4	6.4	2.6	1.3
SelfExtend	128K	✗	100.0	100.0	0.2	19.7	8.6	14.7	22.6	38.0
Infinite	8K	✓	6.8	7.6	0.2	41.5	14.6	20.8	20.6	16.0
Streaming	8K	✓	8.5	8.3	0.4	40.6	14.3	20.4	21.4	16.3
H2O	8K	✓	2.5	2.4	0.0	0.0	0.7	2.8	6.0	2.1
InfLLM	8K	✓	100.0	99.0	5.0	43.7	19.5	24.3	23.7	45.0

serve that: (1) Compared to models with the sliding window mechanism, which can also read extremely long sequences, our method demonstrates a significant performance improvement. This indicates that the context memory in InfLLM can accurately supplement LLMs with relevant contextual information, enabling efficient and effective understanding and reasoning on long sequences. (2) The position downscaling and resuing methods, NTK and SelfExtend, tend to compromise model performance while extending the sequence length to 128K. That is because these models cannot address the distraction issue caused by noisy contexts. In contrast, our model can consistently enhance performance for extremely long sequences. We successfully generalize Llama-3 from a 8K length to more than 16 times its length, achieving commendable performance on the ∞-Bench. (3) The position downscaling and resuing methods can increase the maximum sequence length of LLMs but also raise the computational and memory costs, limiting these methods' application. In contrast, InfLLM utilizes block-level memory and offloading mechanism, enabling efficient processing of long sequences within limited resources.

3.3. Scaling to 1,024K Context

To assess the effectiveness of InfLLM on extremely long sequences, in this subsection, we scale the sequence length to 1024K to evaluate the capacity of InfLLM to capture contextual relevance in long sequences. The results are shown in Figure 2. From the results, we can observe that InfLLM can accurately locate the key information from length noises and achieve 100% accuracy even when the context length scales to 1024 thousand tokens. However, LM-Infinite can only attend to the tokens within the local window, which leads to a rapid decline in its performance

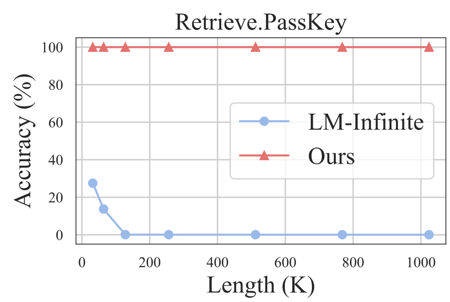

Figure 2: The results on sequences with different lengths.

as the sequence length increases. It proves that InfLLM can accurately capture the long-distance dependencies for effective long-sequence reasoning.

4. Conclusion

In this paper, we propose a training-free method to improve the length generalizability of LLMs. Based on the sliding window attention mechanism, we construct an additional context memory module, which can help LLMs select relevant information from massive contexts to capture long-distance dependencies. The experiments on two widely-used long-text benchmarks show that InfLLM can effectively improve the ability of LLMs, which are trained on sequences with a few thousand tokens, to process extremely long sequences. In the future, we will explore efficient training of the context memory module to further enhance the model performance. Besides, combining the key-value cache compression methods with InfLLM can further reduce the computational and memory costs. We hope InfLLM can boost the development of streaming applications of LLMs.

References

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., and Li, J. Longbench: A bilingual, multitask benchmark for long context understanding. *CoRR*, abs/2308.14508, 2023.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N. S., Chen, A. S., Creel, K., Davis, J. Q., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N. D., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D. E., Hong, J., Hsu, K., Huang, J., Icard, T., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P. W., Krass, M. S., Krishna, R., Kuditipudi, R., and et al. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258, 2021.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *Proceedings of NeurIPS*, 2020.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. Vicuna: An open-source chatbot impressing GPT-4 with 90% ChatGPT quality, March 2023.

Dai, Z., Lai, G., Yang, Y., and Le, Q. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. In *Proceedings of NeurIPS*, 2020.

Ding, Y., Zhang, L. L., Zhang, C., Xu, Y., Shang, N., Xu, J., Yang, F., and Yang, M. Longrope: Extending LLM context window beyond 2 million tokens. *CoRR*, abs/2402.13753, 2024. doi: 10.48550/ARXIV.2402.13753.

Driess, D., Xia, F., Sajjadi, M. S. M., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang, W., Chebotar, Y., Sermanet, P., Duckworth, D., Levine, S., Vanhoucke, V., Hausman, K., Toussaint, M., Greff, K., Zeng, A., Mordatch, I., and Florence, P. Palm-e: An embodied multimodal language model. In *Proceedings of ICML*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8469–8488. PMLR, 2023.

Goyal, S., Choudhury, A. R., Raje, S., Chakaravarthy, V. T., Sabharwal, Y., and Verma, A. Power-bert: Accelerating BERT inference via progressive word-vector elimination. In *Proceedings of ICML*, volume 119, pp. 3690–3699, 2020.

Han, C., Wang, Q., Xiong, W., Chen, Y., Ji, H., and Wang, S. Lm-infinite: Simple on-the-fly length generalization for large language models. *CoRR*, abs/2308.16137, 2023.

Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., Qiu, J., Yao, Y., Zhang, A., Zhang, L., Han, W., Huang, M., Jin, Q., Lan, Y., Liu, Y., Liu, Z., Lu, Z., Qiu, X., Song, R., Tang, J., Wen, J., Yuan, J., Zhao, W. X., and Zhu, J. Pre-trained models: Past, present and future. *AI Open*, 2: 225–250, 2021.

Jin, H., Han, X., Yang, J., Jiang, Z., Liu, Z., Chang, C., Chen, H., and Hu, X. LLM maybe longlm: Self-extend LLM context window without tuning. *CoRR*, abs/2401.01325, 2024.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In Flinn, J., Seltzer, M. I., Druschel, P., Kaufmann, A., and Mace, J. (eds.), *Proceedings of SOSP*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165.

Lewis, P. S. H., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of NeurIPS*, 2020.

Li, D., Shao, R., Xie, A., Sheng, Y., Zheng, L., Gonzalez, J. E., Stoica, I., Ma, X., , and Zhang, H. How long can open-source llms truly promise on context length?, June 2023.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. Code as policies: Language model programs for embodied control. In *Proceedings of ICRA*, pp. 9493–9500. IEEE, 2023.

Lin, S., Asai, A., Li, M., Oguz, B., Lin, J., Mehdad, Y., Yih, W., and Chen, X. How to train your dragon: Diverse augmentation towards generalizable dense retrieval. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of ACL: EMNLP*, pp. 6385–6400. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.423.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *CoRR*, abs/2307.03172, 2023.

LocalLLaMA. Ntk-aware scaled rope allows llama models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation. 2023.

Meta. Introducing meta llama 3: The most capable openly available llm to date. 2024.

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. MTEB: massive text embedding benchmark. In Vlachos, A. and Augenstein, I. (eds.), *Proceedings of EACL*, pp. 2006–2029. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EACL-MAIN.148.

Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., and Schulman, J. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332, 2021.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. In *Proceedings of NeurIPS*, 2022.

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior. In *Proceedings of UIST*, pp. 2:1–2:22. ACM, 2023.

Press, O., Smith, N. A., and Lewis, M. Train short, test long: Attention with linear biases enables input length extrapolation. In *Proceedings of ICLR*. OpenReview.net, 2022.

Qian, C., Cong, X., Yang, C., Chen, W., Su, Y., Xu, J., Liu, Z., and Sun, M. Communicative agents for software development. *CoRR*, abs/2307.07924, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

Tworkowski, S., Staniszewski, K., Pacek, M., Wu, Y., Michalewski, H., and Milos, P. Focused transformer: Contrastive training for context scaling. *CoRR*, abs/2307.03170, 2023.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen, J. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6): 186345, 2024a. doi: 10.1007/S11704-024-40231-1.

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. Improving text embeddings with large language models. *CoRR*, abs/2401.00368, 2024b. doi: 10.48550/ARXIV.2401.00368.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. *CoRR*, abs/2309.17453, 2023.

Xiong, W., Liu, J., Molybog, I., Zhang, H., Bhargava, P., Hou, R., Martin, L., Rungta, R., Sankararaman, K. A., Oguz, B., Khabsa, M., Fang, H., Mehdad, Y., Narang, S., Malik, K., Fan, A., Bhosale, S., Edunov, S., Lewis, M., Wang, S., and Ma, H. Effective long-context scaling of foundation models. *CoRR*, abs/2309.16039, 2023.

Xu, P., Ping, W., Wu, X., McAfee, L., Zhu, C., Liu, Z., Subramanian, S., Bakhturina, E., Shoeybi, M., and Catanzaro, B. Retrieval meets long context large language models. *CoRR*, abs/2310.03025, 2023.

Zhang, X., Chen, Y., Hu, S., Wu, Q., Chen, J., Xu, Z., Dai, Z., Han, X., Wang, S., Liu, Z., and Sun, M. Infinitebench: 128k long-context benchmark for language models, 2023a.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C. W., Wang, Z., and Chen, B. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. *CoRR*, abs/2306.14048, 2023b.

Zhao, L., Feng, X., Feng, X., Qin, B., and Liu, T. Length extrapolation of transformers: A survey from the perspective of position encoding. *CoRR*, abs/2312.17044, 2023.

Broader Impact

This paper presents work whose goal is to advance the field of long sequence processing for large language models. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Limitations

In this paper, we propose InfLLM, a method for extending the context window of LLMs without additional training. We verify the effectiveness of our model using a widely-used long-text evaluation benchmark ∞-Bench. However, our method still has the following limitations: (1) We store a large amount of past key-value (KV) cache in the CPU memory, which increases CPU memory usage. In the future, we can reduce CPU memory requirements by integrating techniques like KV cache quantization. (2) While InfLLM reduces the computational overhead for processing long texts in LLMs, there is still room for speed-up. In the future, we can further enhance the inference speed of InfLLM by integrating it with inference frameworks like llama.cpp[1] and vllm (Kwon et al., 2023).

A. Comparing to Models with Continual Training

In this paper, we focus on expanding the context window of LLMs without additional training. In this section, we compare InfLLM with models that undergo continual training on long sequences in terms of both performance and efficiency. Specifically, we select Llama-3-8B-Instruct-Gradient-1048k (Llama-1M)[2], which has been further fine-tuned on long-text data and chat datasets, extending its context window to 1048K. Besides, we also employ InfLLM on the Llama-1M, where we set the local window as 4K and selected memory size as 4K.

Table 2: The comparison between InfLLM and Llama-3-8B-Instruct-Gradient-1048k (Llama-1M), which is further fine-tuned on long sequences. InfLLM can achieve comparable performance with Llama-1M with less computation consumption and memory usage.

	Train-Free	R.PK	R.Num	R.KV	Choice	QA	Sum	Math.F	VRAM	Time
Llama-1M	✗	**100.0**	**99.8**	**23.2**	**51.5**	13.6	18.5	18.3	76.6G	40.4s
InfLLM	✓	**100.0**	99.0	5.0	43.7	**19.5**	**24.3**	**23.7**	**26.3**G	**26.7**s
Llama-1M+InfLLM	✗	100.0	100.0	55.8	39.3	20.3	17.1	31.4	26.3G	26.7s

We present the results on ∞-Bench, the GPU memory usage, and time consumption in Table 2. From the results, we can observe that: (1) Compared to models that have undergone continual training on long sequences, InfLLM can achieve comparable or even superior results without any additional training. This suggests that LLMs inherently possess the capability to identify key information in long sequences and to understand and reason effectively. Notably, Llama-1M requires 512 GPUs for continual training, which is unaffordable for many researchers. In contrast, InfLLM does not require any training, which indicates the practicability of InfLLM. (2) In terms of efficiency, InfLLM achieves a 34% decrease in time consumption while using only 34% of the GPU memory compared to the full-attention models. Moreover, at longer sequence lengths of 256K tokens, the full-attention baseline fails due to out-of-memory errors, while InfLLM can efficiently process sequences up to 1024K tokens on a single GPU. (3) InfLLM can also be directly combined with the model with continual training and achieve comparable or even superior results with only 8K context window. It indicates that InfLLM can also serve as an efficient way to improve the inference speed.

B. Comparing to Retrieval-Augmented Generation

InfLLM leverages the intrinsic capacity of LLMs to construct a context memory for gathering token-relevant information, a concept similar to retrieval augmented generation (RAG) (Lewis et al., 2020; Nakano et al., 2021). However, compared to using RAG, where historical contexts are treated as a searchable database for long-sequence understanding (Xu et al., 2023), InfLLM has several advantages: (1) Training-Free: RAG requires additional retrieval data to train a retrieval model, whereas InfLLM is training-free and applicable to any LLMs. Besides, RAG also necessitates fine-tuning LLMs to adapt

[1] https://github.com/ggerganov/llama.cpp
[2] https://huggingface.co/gradientai/Llama-3-8B-Instruct-Gradient-1048k

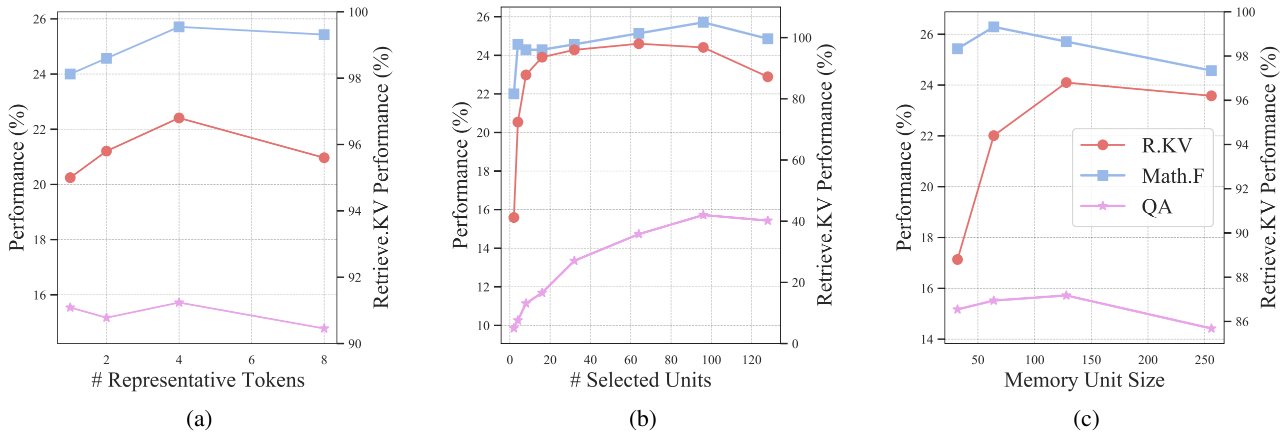

Figure 3: Extra studies about InfLLM. Here, (a), (b), and (c) investigate the impact of the context memory under different numbers of representative tokens, different numbers of selected units, and memory unit sizes, respectively.

to the inputs augmented by the retrieved knowledge. (2) Broader Applicability: RAG models are usually limited by the performance of their retrieval components. Besides, existing retrieval models will suffer from out-of-distribution issues, struggling to perform well on tasks outside their training distribution (Lin et al., 2023; Muennighoff et al., 2023). This limitation adversely affects the overall performance of the RAG system. In contrast, InfLLM has no specific requirements for tasks and can be feasibly used for long sequences.

To verify the generalization capabilities of InfLLM, we conduct experiments to comparing RAG and InfLLM on three context retrieval tasks. We utilize E5-mistral-7B-instruct (Wang et al., 2024b) as the retrieval model. The results are shown in Table 3. Our findings demonstrate that even without additional data or training, InfLLM can consistently outperform RAG models, underscoring its superior generalization capabilities. The dependency on an external retrieval model makes RAG less flexible in handling diverse tasks.

Table 3: The comparison between InfLLM and RAG.

Task	R.PK	R.Num	R.KV
RAG-E5	89.2	65.4	13.2
InfLLM	**100.0**	**96.1**	**96.8**

C. The Impact of Memory Settings

InfLLM relies on the context memory to look up relevant information. We further explore the impact of core components in the context memory, specifically the representative tokens and memory units. The results are shown in Figure 3.

Different Number of Representative Tokens. InfLLM splits key-value vectors into memory units and selects several representative tokens from the unit to serve as the unit representations. Consequently, the ability of these representative tokens to semantically represent the entire unit directly impacts the model's performance. We conduct experiments with the number of representative tokens as $\{1, 2, 4, 8\}$. The results are shown in Figure 3(a). It is observed that as the number of representative tokens increases, there is a trend of improvement in the model performance, which indicates that more representative tokens tend to better represent the semantic content of the memory units. However, it is noted that when the number of representative tokens reaches 8, there is a slight performance decrease. This decline can be attributed to the inclusion of semantically irrelevant tokens as unit representations. More efficient and powerful unit representations will further enhance model performance for future work.

Different Number of Selected Units. The selected units are utilized to provide relevant context to LLMs. We conduct experiments with the number of units set as $\{2, 4, 8, 16, 32, 64, 96, 128\}$. From Figire 3(b), we can observe that as the number of selected units increases from 1 to 32, the model performance significantly improves, which is attributed to that more units imply a greater recall rate of relevant content. Larger unit quantity also leads to an increase in the required memory scheduling time and the computational time for attention. Therefore, further enhancing lookup accuracy remains a crucial direction for improving the efficiency of InfLLM.

Different Memory Unit Size. Each memory unit is supposed to be a coherent semantic unit. Excessively large unit sizes can hinder precise lookup, while a small size will increase the computational overhead of memory lookup. We evaluate InfLLM with the unit size as $\{32, 64, 128, 256\}$ and keep the total context length as 12K. The results are shown in Figure 3(c). It can be observed that the optimal unit size varies for different tasks due to the varying characteristics of input sequences. For example, in Retrieve.KV, a key-value pair constitutes a semantic unit, while in Math.Find, a single number represents a semantic unit. Employing heuristic rules to segment context can easily lead to suboptimal performance. Therefore, exploring how to dynamically segment context is an important direction for future research.

D. Cache Management Strategy

Due to the massive amount of memory units for extremely long sequences, we adopt an offloading mechanism to save GPU memory costs. Considering the infrequent usage of memory units, we offload most memory units to CPU memory and only preserve the frequently used memory units and current needed memory units in the GPU memory. To this end, we maintain a cache in GPU memory to effectively utilize GPU memory and reduce the communication between CPU and GPU. The size for our GPU cache is fixed, and therefore we design a least recently used (LRU) strategy for cache management. In this section, we will introduce the management strategy in detail.

Loading Memory Units For each computation step, we first compute the relevance scores for each memory unit to determine which units should be used. Then, for each needed memory unit, we first search it in our cache. If there is no hit, then we proceed with the transfer from CPU memory to GPU memory.

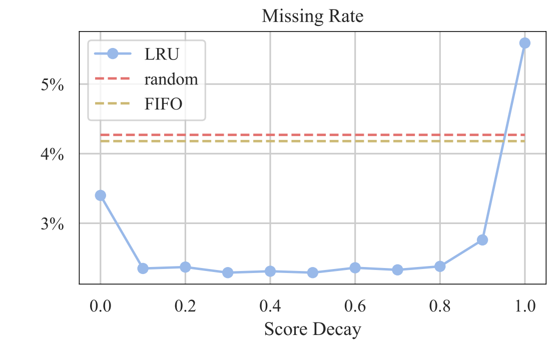

Figure 4: Missing rates of different cache management strategies.

Offloading Memory Units After the attention computation, we need to offload redundant memory units to keep the GPU cache fixed. To this end, we apply an LRU strategy. Specifically, for each memory unit loaded into our GPU cache, we assign a frequency score s_b for it, which will be used to determine whether this unit should be maintained in the GPU cache or offloaded to CPU memory to save GPU memory costs. The frequency scores are updated after the attention computation. Specifically, we update the score as follows:

$$s_b = s_b \cdot d + \sum_{j=1}^{l_X} \sum_{i=1}^{l_{bs}} \text{attention\_score}(\mathbf{q}_{j+l_P}, \mathbf{k}_i), \tag{1}$$

where l_u represents the number of current tokens involved in this lookup, attention_score$(\mathbf{q}, \mathbf{k})$ denotes the attention score between $\mathbf{Q}$ with respect to $\mathbf{k}$ (ranging from 0 to 1) obtained after performing the attention computation. d is a hyper-parameter, representing the decay coefficient, used to incorporate the influence of previous lookups. After each attention computation, we sort all the memory units in our GPU cache according to their frequency scores s_b, and offload the units with the lowest scores back to the CPU memory.

To verify the effectiveness of our cache management strategy, we evaluate the cache missing rate of different cache management strategies on a sample of data from the GovReport dataset. Specifically, we compare our LRU strategy with (1) Random: randomly selecting units from the GPU cache to offload. (2) First-in-first-out (FIFO): offload the unit that is first loaded in the GPU cache. The results are illustrated in Figure 4. It is observable that the LRU strategy we employed exhibits a lower missing rate, which ensures that the offloading mechanism does not introduce significant time overhead. In the experiments described in the main text, we chose a decay value of 0.1.

E. Positional Encoding

In InfLLM, we assign all tokens beyond the local window size with the same positional encoding. Therefore, for the current tokens, we do not explicitly provide positional information for the context. But we think that the unidirectional nature of a decoder-only model allows it to recognize the positional information of the context. For instance, assume a sequence contains three spans S_A, S_B, and S_C in order. When encoding S_C, although S_A and S_B are assigned the same positional encoding, the unidirectional nature of the decoder-only model allows the key-value hidden states of S_A and S_B inherently embeds their relative positional information: S_B can utilize information from S_A during its encoding, while S_A can only access information from preceding parts of the sequence.

Table 4: The results of InfLLM and baseline models on LongBench. The 95% quantile for text lengths in LongBench is 31K. The context window size for sliding window models refers to the local window size, and for InfLLM refers to "local window size + selected memory size".

	Window	NQA	Qasper	MFQA	HQA	2WikiMQA	Musique
			Mistral-based Models (7B)				
Mistral	32K	22.06	29.16	47.65	37.53	21.96	19.03
Infinite	6K	18.44	30.02	39.05	32.02	22.27	15.81
Streaming	6K	17.92	30.05	39.09	32.18	21.83	14.71
InfLLM	6K	22.12	29.33	47.42	36.56	22.31	17.68
InfLLM	12K	23.03	29.52	47.62	39.53	23.61	18.92
			Llama-3-based Models (8B)				
Llama-3	8K	19.85	42.36	41.03	47.38	39.20	22.96
Infinite	8K	19.39	42.80	40.44	43.77	37.89	18.33
Streaming	8K	20.05	42.46	39.54	43.69	37.89	19.68
InfLLM	8K	22.64	43.70	49.03	49.04	35.61	26.06

	Window	GovReport	QMSum	MultiNews	TREC	TQA	SAMSum
			Mistral-based Models (7B)				
Mistral	32K	31.12	23.87	26.62	71.00	85.97	42.29
Infinite	6K	29.74	21.92	26.65	70.00	85.22	41.60
Streaming	6K	29.83	21.94	26.64	70.00	85.57	41.31
InfLLM	6K	31.03	23.49	26.70	69.00	86.67	42.52
InfLLM	12K	31.37	23.77	26.66	71.00	87.34	41.80
			Llama-3-based Models (8B)				
Llama-3	8K	29.94	21.45	27.51	74.00	90.50	42.30
Infinite	8K	29.25	21.41	27.62	74.00	90.08	41.72
Streaming	8K	29.17	21.33	27.56	73.50	90.08	41.55
InfLLM	8K	30.76	22.70	27.57	73.50	90.91	42.43

	Window	PsgCount	PsgRetrieval	LCC	RepoBench-P	Avg.
			Mistral-based Models (7B)			
Mistral	32K	3.95	86.94	57.42	54.14	43.78
Infinite	6K	2.08	42.80	57.12	53.43	39.07
Streaming	6K	2.50	42.17	55.38	51.46	38.67
InfLLM	6K	2.87	64.00	56.67	52.97	41.90
InfLLM	12K	3.01	87.42	56.69	52.09	44.02
			Llama-3-based Models (8B)			
Llama-3	8K	8.50	62.50	60.83	49.14	44.73
Infinite	8K	4.50	50.00	60.12	48.62	43.03
Streaming	8K	5.00	49.00	60.35	48.95	42.99
InfLLM	8K	7.17	84.00	59.88	46.48	46.95

To verify the model's capability to capture the relative positional information of the context, we adopt the Retrieve.Passkey task with multiple pass keys for evaluation. In this task, each sequence contains two pass keys, and the model is required to output these two pass keys in order. The data construction approach is consistent with that of ∞-Bench (Zhang et al., 2023a), where the positions of the two pass keys are randomly selected. We created 50 sequences, each 64K in length. The experimental results reveal that in this task, InfLLM can output the values of the two pass keys in the correct order 100% of the time. This indicates that, although our positional encoding disregards the relative positional information of the context, the model can still effectively understand the context in sequence.

F. External Experiments

F.1. Implementation Details

The context memory is constructed for all layers in LLMs. We set the size of our GPU cache as 32, which is twice the number of loaded units for each step. We set the frequency score decay coefficient as 0.1. We adopt the half-float precision for all experiments. We use NVIDIA A100 or A800 to conduct our experiments. For the experiment that scales to $1,024$K context, we set the encoding chunk size as 2048, and the number of representative tokens as 1 to speed up experiments.

F.2. Performance on LongBench

We also employ LongBench (Bai et al., 2023) as the benchmark to evaluate the effectiveness of InfLLM and baseline models. The evaluation results are shown in Table 4. The results indicate that: (1) InfLLM outperforms other models capable of processing streaming inputs across various diverse tasks. It proves that the context information provided by the context memory can efficiently enhance the model performance. (2) When applying Llama-3 as the base model, both StreamingLLM and LM-Infinite achieve only comparable or even worse performance than the original Llama-3. This indicates that while sliding window attention can effectively extend the context window size of LLMs, these models discard long-distance contextual information, thereby failing to achieve effective long-sequence understanding. (3) Mistral can handle text lengths up to 32K, covering most instances in LongBench. In contrast, InfLLM, with a window size of only 12K, achieves comparable or even superior performance on average. This further demonstrates InfLLM's ability to filter out noise in long contexts, leading to better long-sequence understanding.

F.3. Experiments on Vicuna

In the previous sections, we demonstrated that InfLLM can extend the context windows of Llama-3 (with a maximum length of 8K) and Mistral (with a maximum length of 32K) to several hundred thousand tokens. To further validate the effectiveness of InfLLM, we apply it to the Vicuna (Chiang et al., 2023), which has a maximum length of only 4K. The experimental results are shown in Table 5. The results show that we effectively extend Vicuna's context length to 128K, achieving significant performance improvements on the Retrieve.Passkey and Retrieve.Number tasks. However, InfLLM can not show performance gains on the Retrieve.KV and Math.Find tasks. This is because the hidden vectors contained in Vicuna have a limited ability to filter out noise in extremely long texts, making it difficult for context memory to effectively locate relevant information in the more complex contexts of the Retrieve.KV and Math.Find tasks. In the future, It deserves further exploration to design more powerful memory mechanism.

Table 5: The results of Vicuna-based models.

	R.PK	R.Num	R.KV	Math.F
Vicuna	5.08	4.41	1.40	11.71
InfLLM	99.15	81.69	0.60	11.14