Extending Context-Awareness in StreamingLLM with Retrieval-Augmented Generation

Miguel S. Moreira
miguelsm@mit.edu
922051799

Luc Dao
daoluc@mit.edu
916696988

Jaehyun Sim
simjay@mit.edu
975935556

Christopher McNally
mcnallyc@mit.edu
920969355

Abstract

Large language models (LLMs) excel in many language tasks but are constrained by limited context windows, hindering their performance on long-sequence inputs such as event streams or knowledge databases [1, 2]. Although recent approaches have addressed these limitations through improved attention mechanisms, sparsity, and pruning [3, 4, 5, 6], context length remains inherently bounded by kernel memory. In particular, StreamingLLM [3] achieves high performance even on long-sequence inputs, but nevertheless cannot retrieve information outside its rolling context window. This work leverages retrieval-augmented generation (RAG) to integrate external memory mechanisms, enabling virtually infinite textual coherence in addition to facilitating access to historical data and external databases. We further explore this solution's advantages, limitations, and trade-offs compared to a baseline StreamingLLM system. Needle-In-A-Haystack-style experiments show that RAG effectively extends context-awareness in StreamingLLM. Our implementation, data, results, visualizations, and demo video are available at `https://github.com/miguelsmoreira/streaming-llm`.

1 Introduction

Large Language Models (LLMs) are increasingly used in applications requiring long, uninterrupted text processing, such as multi-turn dialogues and document analysis. However, their performance is limited by pre-training attention windows. With their growing adoption in commercial (personal assistants) [7], industrial (process control) [8], and medical (imaging, laboratory or procedural examination analysis) [9] contexts, where requirements for accuracy and performance are stringent, and failures can have severe consequences, there is a pressing need to develop techniques capable of handling near-infinite text sequences efficiently.

The increasing context lengths of recent models, summarized in table 1, highlight their critical importance in achieving state-of-the-art performance. However, the fundamental $O(n^2)$ compute scaling of the attention mechanism means that hardware memory limitations constrain further advances. A new approach will be needed.

We explore and develop long-term memory mechanisms inspired by retrieval-augmented generation (RAG) to dynamically integrate evicted information. This approach not only enables coherent processing of near-infinite text sequences but also facilitates access to historical data and external databases. While such mechanisms may introduce latency due to external memory access and relevance computation, we expect these trade-offs to be justified in many applications—particularly in critical domains such as healthcare—where the added context could significantly enhance outcomes. Furthermore, optimizations such as prefetching, caching, or heuristic gating could mitigate these latencies by predicting when additional context is needed and streamlining retrieval processes.

Preprint. Under review.

This project integrates RAG techniques with StreamingLLM [3] to restore the model's ability to access evicted information. As a demonstration, we develop a chatbot capable of processing long-context information, a capability directly applicable to a variety of relevant scenarios, from personal assistants to news-casting and large databases.

Model	Context Window	Quality	Throughput (tokens/s)	Latency (s)
Gemini 1.5 Pro	2M	80	60.0	0.75
Gemini 1.5 Flash	1M	73	249.7	0.34
Jamba 1.5 Mini	256k	64	82.6	0.49
Jamba Instruct	256k	28	76.0	0.52
Claude 3.5 Sonnet	200k	80	54.7	0.88
Claude 3 Opus	200k	70	26.3	1.99
GPT-o1	128k	85	33.3	30.50
GPT-4o	128k	77	82.3	0.42
Llama 3.1 405B	128k	72	72.6	0.88
Llama 3.2 3B	128k	47	201.8	0.34

Table 1: Comparison of large language models across context window size, quality index, median throughput, and median latency. Quality index is a composite metric based on a mixture of benchmarks (higher is better). Data reproduced from [10].

2 Background

2.1 KV Cache Optimization Techniques

Over the years, numerous techniques have been developed to address the challenges of processing long text sequences with LLMs [11, 12, 13, 14, 15, 16, 17]. The shift from dense attention to windowed attention allowed for longer sequences. However, recall degrades when initial tokens are *evicted* from the window. Enhancements like windowed attention with recomputation prevented complete breakdowns but remain susceptible to information loss due to truncation of earlier tokens.

Recent advances have focused on optimizing limited memory to maximize "effective" context lengths. One notable development is the concept of attention sinks, introduced in StreamingLLM [3]. Attention sinks refer to the retention of the key-value (KV) cache for the initial tokens, which naturally receive a high concentration of attention across all layers of the model. This mechanism prevents performance degradation when earlier tokens are evicted from the cache. Although effective for stabilizing performance, attention sinks cannot access evicted tokens, limiting their ability to provide true long-term memory. This limitation echoes challenges in recurrent architectures, such as RNNs and LSTMs, where information is progressively lost over extended sequences.

Building on these ideas, hierarchical attention mechanisms have been explored to separate global and local attention. Sparse attention is applied to global features, while dense attention—supported by attention sinks—is reserved for local features. DuoAttention [4] builds on this approach by using different-sized KV caches for global and local features, improving memory efficiency and extending the utility of attention mechanisms in long-context settings.

Finally, techniques like query-aware sparsity [6] aim to accelerate inference at long-context lengths by approximating attention scores within paged KV cache blocks rather than at a per-token level. While these methods do not prevent token eviction when memory bounds are exceeded, they should significantly optimize inference performance in systems with large memory capacities by dynamically reducing computational overhead, providing another path to efficient handling of long-context inputs within the model's predefined attention window.

2.2 Retrieval-Augmented Generation

While there has been an explosion of research in efficient transformer architectures, LLMs remain fundamentally constrained by limited memory in inference and training systems. For example, a 100,000-GPU H100 cluster provides approximately 16 PB of HBM memory, compared to exabyte-scale storage commonly available in traditional high-performance computing (HPC) centers, using

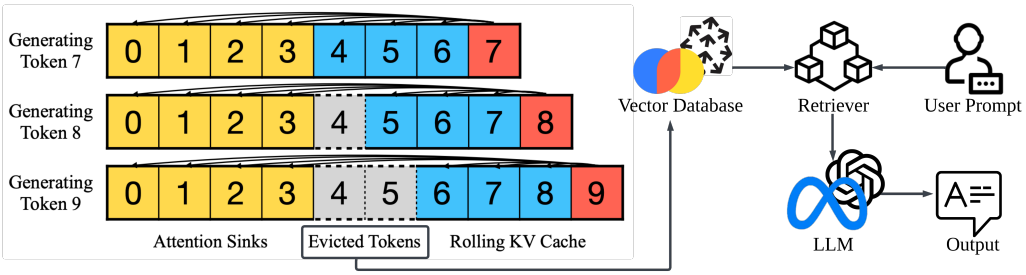

Figure 1: System diagram of RAG-augmented StreamingLLM. Image based on figure from [3].

hard drives or tape. To overcome this limitation, one can leverage external storage systems that integrate additional context directly in the incoming prompt, doing away with the need for fundamental changes to the underlying architectures commonly used in inference.

Retrieval-Augmented Generation (RAG) is a framework that integrates information retrieval into the generation process. This approach addresses challenges like knowledge updating, handling long-tail data, and reducing training and inference costs. In a typical RAG system, a retriever identifies relevant data based on a user query, and a generator, such as an LLM model, incorporates this data to produce richer, more context-aware content. By serving as a form of non-parametric memory, RAG systems allow for easy updates, support extensive knowledge coverage, reduce reliance on large parameterized models, and partially separate the concerns of information storage and novel text generation. This makes RAG especially valuable for applications like question answering, code generation, and image generation [18].

In the context of this project, we leverage RAG as a framework to integrate evicted tokens back to StreamingLLM to ensure the coherence and relevance of infinite sequence lengths. The evicted tokens are stored in a vector database. When a new user prompt comes in, the relevant context will be retrieved from the database and added to the prompt before sending to StreamingLLM for output generation (see fig. 1).

3 Implementation

Our retrieval system consists of three primary components that work in tandem with StreamingLLM:

1. A context window that mirrors StreamingLLM's KV cache size
2. A lightweight embedding model (`all-MiniLM-L6-v2`)
3. An in-memory vector database (ChromaDB)

The system is designed to run entirely locally on personal hardware, eliminating the need for external services or cloud infrastructure. This is achieved by selecting lightweight, efficient components: ChromaDB for vector storage and `all-MiniLM-L6-v2` for generating 384-dimensional embeddings, both of which have minimal computational requirements. For the base LLM, we used `meta-llama/Llama-2-7b-chat-hf`, which offered an adequate tradeoff between generation quality and performance on consumer-grade hardware.

When new prompts arrive in the system, the retriever first queries the vector database to identify the relevant historical context. This search process utilizes the similarity of the ℓ^2-norm in the embedding space to find the most relevant context that could inform the current prompt. The retrieved content is prepended to the incoming prompt before being processed by StreamingLLM.

As the system processes inputs and generates outputs, both the prompts and the resulting outputs are added to the context window. When the context window grows beyond the size of the StreamingLLM KV cache, the system automatically manages this overflow by evicting the oldest tokens and transferring them to the vector database for long-term storage and future retrieval (see fig. 2).

This synchronized operation ensures that the context window mirrors the current state of StreamingLLM's KV cache while the vector database serves as a repository for historical information

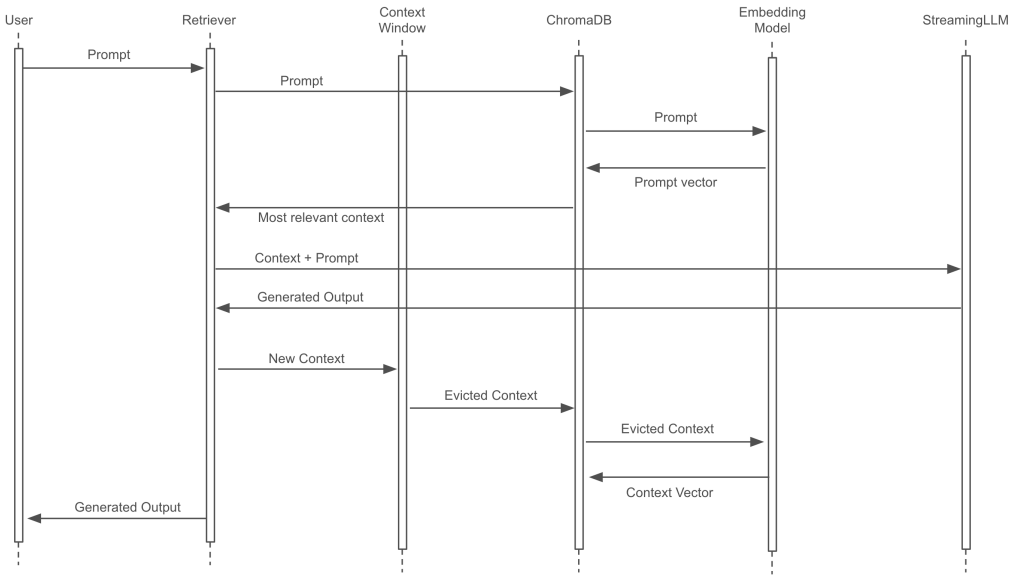

Figure 2: Workflow of StreamingLLM with retriever, evicted-only mode

that has been evicted from active memory. As a result, the system can maintain both immediate context awareness and access to relevant historical information.

To explore the impact of various setups on Retrieval-Augmented Generation, the system supports two retrieval modes for experimental purposes: *evicted-only mode*, where the retriever only accesses prompts that have been evicted from the KV cache (see fig. 2), and *always-on mode*, where the retriever can access all historical prompts including the present one, regardless of their eviction status. Always-on mode is in some sense *acausal*: the retriever can bring information into context that has not yet been processed by StreamingLLM. This flexibility enables comparative analysis of StreamingLLM's RAG performance under different retrieval strategies.

4 Experiments and Results

To evaluate the RAG-augmented StreamingLLM model, we prepare a family of documents containing a target phrase that is unlikely to be in the training data. (The particular phrase used for the tests was, "*The best thing to do in San Francisco is eat a sandwich and sit in Dolores Park on a sunny day.*") These documents range in length from 1,000 to 10,000 tokens. The keyword's location ranges from the beginning of the document (0%) to the end (100%). The rest of the documents are filled with irrelevant but sensible English text. Following the precedent of [19], the document lengths are referred to as *context lengths*, but we note that this is potentially misleading terminology; it has no relationship with the model context length.

For each document, we initiated a chat interaction with either the base StreamingLLM system, an RAG-augmented StreamingLLM as described above, or an always-on RAG-augmented model that stores the entire prompt in the vector database before prompting. After sending the document, we prompt the assistant with a question regarding the target phrase: "*What is the best thing to do in San Francisco? Don't give information outside the document or repeat your findings*"

The assistant's response is then compared with the target phrase using one of two measures of similarity. The first is a cosine-similarity-based semantic distance measure. We embed the output o and target phrase t with the embedding function $\mathbf{e} = \texttt{all-MiniLM-L6-v2}$, obtaining embedding vectors $\mathbf{e}(o)$ and $\mathbf{e}(t)$. Then we return

$$\texttt{SemanticSimilarity}(o,t) = \frac{1}{2} + \frac{\mathbf{e}(o)\cdot\mathbf{e}(t)}{2||\mathbf{e}(o)||||\mathbf{e}(t)||}, \tag{1}$$

the cosine distance scaled and shifted to have codomain $[0,1] \subseteq \mathbb{R}$.

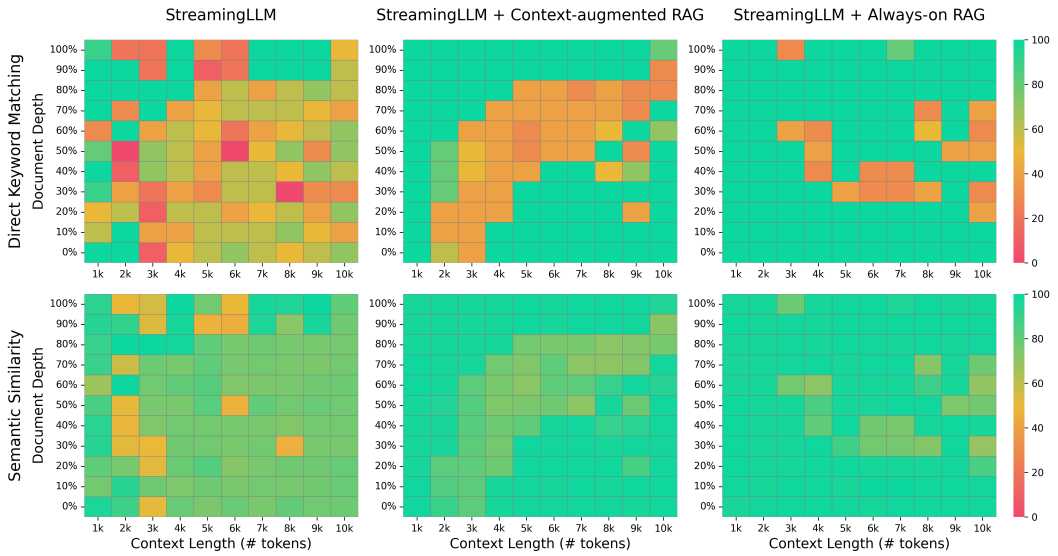

Figure 3: Probability of keyword retrieval as a function of test document length and keyword position. **(Above)** Measure of similarity based on keyword matching heuristic. **(Below)** Measure of similarity based on semantic distance. **(Left)** Base StreamingLLM system with Llama 2. **(Middle)** Augmented with evicted-only mode RAG. **(Right)** Augmented with always-on RAG.

The distance measure `SemanticSimilarity` is motivated by the desire to preserve the meaning of the target phrase while tolerating differences in wording. However, it has some limitations. In particular, the meaning of its range is rather obscure: entirely unrelated phrases will attain values only slightly less than 1, so it is difficult to interpret as an "accuracy" between 0 and 1 in any meaningful sense.

For a more interpretable measure of the output accuracy, we also use a *direct keyword matching* distance measure, which tests the output for 10 keywords appearing in the target phrase and returns the fraction of them that are present. The tradeoff is that this distance measure is entirely *ad hoc*, and doesn't naturally scale even to experiments with multiple target phrases.

Figure 3 depicts the accuracies of the systems on each document according to each similarity measure and system configuration. The top row shows the direct keyword similarity measure, and the bottom shows the cosine-similarity-based measure. Columns correspond to different system configurations: on the left, we report the results for the base StreamingLLM system. On the right, the RAG-augmented StreamingLLM is shown as described in section 3. In the middle, we show the results for StreamingLLM augmented with always-on RAG.

Examining the first column of fig. 3, we observe that the base StreamingLLM model achieves high accuracies when the absolute position of the target phrase is within ≈ 2000 tokens of the end of the document.

In the second column, we see that the augmented system recovers high accuracy when the target phrase is sufficiently early in the document. However, there remains a 'blind spot' between the recent tokens and those that are retrieved from the vector database. We will return to this in section 5.

The third column shows the same system with always-on RAG, which preemptively stores all prompts into the vector database, regardless of their presence in the KV cache. In principle, this should achieve perfect accuracy regardless of the target-phrase depth or document length; however, it fails in a few examples. Direct examination of the interaction transcripts reveals that these were due to base model refusals rather than failures to retrieve the target phrase.

5 Discussion and Analysis

Limitations include the granularity of the similarity metric and the determinism of our evaluation. Because there was only a single document of each length and target-phrase depth, we resorted to somewhat artificial accuracy scores, neither of which is fully satisfactory. It might be preferable to produce a large number of random documents of each length and target-phrase depth, and average some objective metric over that ensemble.

Instead of a specific English phrase, it may be preferable to use a random string. Unfortunately, Llama 2, our choice of base model, refused to retrieve anything that resembled sensitive data like a password. It is possible that this could be bypassed with inventive prompting, but it is likely that we could benefit from using a different model.

We also attribute the 'blind spot' seen in the middle column of fig. 3 to particularities of the base model. Although `meta-llama/Llama-2-7b-chat-hf` was trained with a context window of 4096 tokens, it seems empirically not to reliably recover information in the context more than ≈ 2000 tokens away, as can be seen in the left column of fig. 3. We expect this limitation to be base-model-dependent, and it may be entirely absent for some choices of base model.

6 Conclusion

This report addresses the limitations of current techniques for processing long text sequences in large language models (LLMs), which remain constrained by hardware memory despite advances like attention sinks and sparse attention. To overcome these challenges, we integrate retrieval-augmented generation (RAG) with StreamingLLM, enabling the dynamic retrieval of evicted information for coherent, near-infinite-context processing. Needle-in-a-haystack-style experiments show that RAG effectively extends context awareness in StreamingLLM.

7 Data and Reproducibility

All tests were run on an Apple M4 Max chip with 128 GB of unified memory. All of our code, data, notebooks, and results can be found in the git repository at `https://github.com/miguelsmoreira/streaming-llm`. The visualization notebook is available in the project repository at `/visualization/heatmap.ipynb`. Prompt documents are available in the project repository at `/data`, and interaction transcripts at `/results`. The commands used to run the model are available in `/results/overnight.txt`.

8 Contributions

The retriever with vector database was implemented by Luc Dao with the support of Christopher McNally. The integration and testing of the model was performed by Miguel Moreira with the support of Jaehyun Sim. Benchmarking was conducted by Miguel Moreira and Jaehyun Sim. The poster presentation materials were prepared by Jaehyun Sim and Miguel Moreira. This final report was written by Christopher McNally and Luc Dao.

We acknowledge the valuable support provided by the Teaching Assistants of the TinyML and Efficient Deep Learning Computing subject, especially Guangxuan Xiao for his insightful clarification of the technical details of StreamingLLM.

References

[1] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[2] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd*

Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[3] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024.

[4] Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads, 2024.

[5] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, February 2021.

[6] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024.

[7] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, Mengwei Xu, Zhijun Li, Peng Li, Yang Liu, Ya-Qin Zhang, and Yunxin Liu. Personal llm agents: Insights and survey about the capability, efficiency and security, 2024.

[8] Mohamad Fakih, Rahul Dharmaji, Yasamin Moghaddas, Gustavo Quiros, Oluwatosin Ogundare, and Mohammad Abdullah Al Faruque. Llm4plc: Harnessing large language models for verifiable programming of plcs in industrial control systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '24, page 192–203. ACM, April 2024.

[9] Minh-Hao Van, Prateek Verma, and Xintao Wu. On large visual language models for medical imaging analysis: An empirical study, 2024.

[10] LLM leaderboard. `https://artificialanalysis.ai/leaderboards/models`. Accessed: 2024-12-14.

[11] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799*, 2019.

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

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

[14] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[15] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[16] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

[17] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

[18] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey. *CoRR*, 2024.

[19] LLMTEST needle in a haystack. `github.com/gkamradt/LLMTest_NeedleInAHaystack`, 2023. Accessed: 2024-12-08.