Retrieval Augmentation for StreamingLLM

Alex Cheng
Harvard University
alexcheng@college.harvard.edu
alexc628@mit.edu

Jeremy Hsu
Harvard University
jeremyhsu@college.harvard.edu
jeremyhs@mit.edu

Kevin Huang
Harvard University
kevin_huang@college.harvard.edu
kyh@mit.edu

Code and Demo: https://github.com/avcheng/RA-StreamingLLM
Presentation: Slides

Abstract

StreamingLLM was created as an efficient framework to enables large language models (LLMs) trained with a finite length attention window to generalize to infinite sequence lengths without fine-tuning. However, one limitation of StreamingLLM is its inability to consider previously evicted tokens. The result is a potential memory lapse of items outside of the current window. In this paper, we use concepts from Retrieval Augmented Generation (RAG) to enable infinite-length content storage to improve StreamingLLM's memory. Through our ad hoc testing, we determine that our RA-StreamingLLM maintains infinite window processing and coherence while enhancing accuracy and relevance relative to StreamingLLM when a user inputs long prompts or a series of prompts.

1 Introduction

Large language models (LLMs) have recently seen a surge in popularity with models being able to achieve human-level performance on academic benchmarks such as the LSAT [1], enabling models to be used across a large swath of different applications such as dialog systems, document summarization, code completion and question answering. However, these models are trained with a finite context w. Currently, LLMs have a finite-length attention window which means that once context has exited the window (i.e., any context that is no longer with the N most recent tokens), it is no longer explicitly memorized by the model. As a result, models have been shown to fail to maintain coherence when conducting long conversations, summarizing long documents, or executing long-term planning.

However, StreamingLLM does not consider tokens which have been evicted from the model's attention window. Therefore, LLMs running with StreamingLLM appear to have short-term memory, in which they cannot factor in earlier conversational exchanges in their current context.

Retrieval Augmented Generation (RAG) was introduced as a strategy for expanding the knowledge of Language Models (LLMs) beyond their initial training data by leveraging external databases. With this project, we aimed to combine RAG and StreamingLLM to produce RA-StreamingLLM to generate more accurate and relevant responses, especially in response to longer prompts and

Preprint. Under review.

conversations, while still being able to process and generate text without any length constraints and maintain coherence.

2 Background

2.1 Retrieval-augmented generation (RAG)

Retrieval-Augmented Generation (RAG) is a concept to optimize the output of a large language model by using a knowledge base outside of the LLM's original training data sources before generating a response [2]. Large Language Models (LLMs) are trained a vast amount of data, however, RAGs can be used for specific domain knowledge, a specific organizations data, or new knowledge that the LLMs have not been trained on. By doing so, RAG is able to extend the capabilities of LLMs without the need to retrain the model.

In RAG, the data is typically loaded and prepared for queries or "indexed". User queries then are then used to query the index, which retrieves the data with the most relevant context. This context and your query then go to the LLM along with a prompt, and the LLM provides a response. However, in our scenario, the data is not preloaded into the index and is added as we evict tokens from the Key-Value Cache.

2.2 Key-Value Cache (KV Cache)

A KV Cache is a specialized memory system for LLMs that stores calculated key-value pairs during the generation process. It allows the LLM to reuse previously computed information instead of recalculating it from scratch for each token it generates, significantly improve the speed and efficiency of generating text. During the generative process, an LLM performs a series of calculations using the context to predict the next token. These calculations involve key-value pairs, which represent the relationships between different parts of the text. The LLM stores them in the KV Cache, which the LLM can check the KV Cache to see if the relevant key-value pairs already exist. If they do, the LLM can reuse them directly, without having to recalculate them.

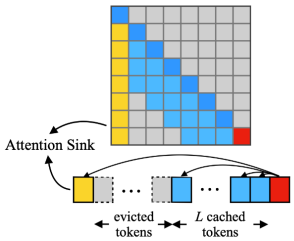

Figure 1: StreamingLLM Key Value cache eviction, Source: [3]

As shown in Figure 1, StreamingLLM keeps the first few KV Cache values, known as the attention sinks, and the most recent KV Cache values up to the allotted size of the cache. All other tokens in between are evicted, maintaining the size of the cache and allowing for efficient computation. These evicted tokens are then forgotten by the LLM, meaning they cannot be used for context for future prompts.

3 RA-StreamingLLM

To integrate RAG with StreamingLLM, we experimented on the LMSYS Vicuña-13b-v1.3 model, a chat assistant trained by fine-tuning LLaMA on user-shared conversations collected from ShareGPT [4] though our method should be generalizable to any autoregressive language model that employs relative positional encoding and enables access to the output of the embedding layer. We first tried a naive approach using prompt engineering by utilizing context retrieved by LlamaIndex's query engine, an open source RAG framework, to ensure this general approach was viable [5]. Then we developed a custom evicted token storage and retrieval using Redis for evicted KV Cache storage and

ChromaDB, an open source embedding database, for vector storage. Both approaches are described below.

3.1 Proof of concept: Prompt Engineering

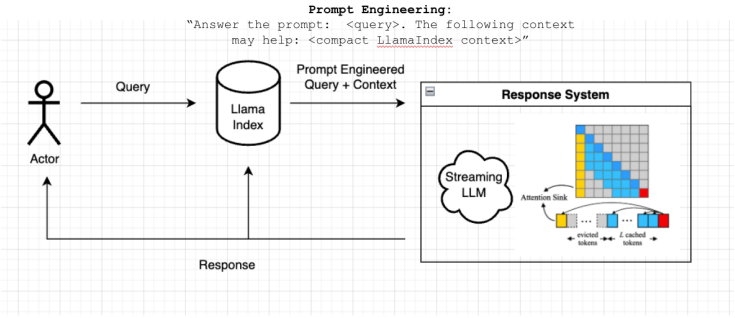

Figure 2: Prompt Engineering using LlamaIndex

As an initial test, we implemented RAG style retrieval using prompt engineering. As shown in Figure 2, for this approach each query was embedded with LlamaIndex and inserted into the index for later retrieval. Then, the LlamaIndex was queried to find the most similar previous embeddings. These embeddings were then used to synthsize context by LlamaIndex, which was then fed into StreamingLLM, along with the origina query in the following format:

```
"Answer the prompt: <query>. The following context may help: <compact
                    LlamaIndex context>"
```

StreamingLLM's response was then fed back to the user, with the response also being embedded and inserted into LlamaIndex. Results from testing this showed promise that a RAG framework would work with StreamingLLM and can be seen in the slides. Through ad-hoc testing, we saw that using this method, the model was sometimes able to provide the correct answers with context from previous prompts, showing promise that this concept would work. However, this method was saw a drastic increase in latency due to the fact that LlamaIndex had to generate text for the context inputted into StreamingLLM, making the system effectively having to query two models. On top of this, the context generated not always correct or relevant. Thus, we turned to creating our own custom evicted token KV retrieval.

3.2 Our implementation of RA-StreamingLLM

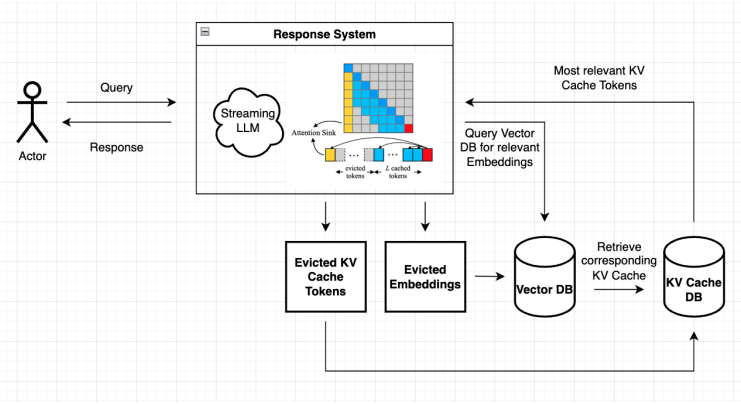

Figure 3: Custom evicted KV Cache Storage and Retrieval

For our implementation of RA-StreamingLLM, we developed a custom evicted KV cache token eviction and retrieval system by utilizing a vector DB using ChromaDB [6] to store the embeddings and Redis DB to store and retrieve the evicted KV Cache.

When a new query was received, if tokens needed to be evicted by StreamingLLM, they were pickled and stored in the Redis database, and their respective embeddings were stored in the vector DB under the same id. Then, the vector DB was queried in order to retrieve the ids of the most similar embeddings, which were then used to get the pickled KV cache tokens in Redis. The tokens were then unpickled and inserted back into the attention window of the LLM.

By directly utilizing KV Cache evicted tokens calculated by Streaming LLM, there was no need to recalculate or regenerate context like our previous method. This allowed for lower latency and more efficient computation.

4 Evaluation

4.1 Initial ad-hoc evaluation

Initially, to test our models, we gave various prompts that contained context, provided a series of unrelated prompts, and then asked the model to retrieve the context from the origianl prompt. Results were mixed, but showed promise that our method could properly retrieve context that was previously evicted.

However, for most of the the initial tests, we only had an M1 Mac which took a long time to generate output, and therefore would take a long time to get to the point where tokens in the KV cache started to be evicted. Thus, we decided to use smaller context window size of 500 tokens and max generation size of 250 tokens to test our systems. This allowed us to test out more prompts in a shorter amount of time, as the KV Cache in StreamingLLM would reach capacity much faster in a smaller number of prompts and responses and start needing to evict tokens.

Results were evaluated for our RA-StreamingLLM by comparing answers to long prompts against StreamingLLM's responses to the same long prompts, using a system with a NVIDIA A100 Tensor Core GPU that we got limited temporary access to. We directly evaluated the model's ability to remember less recent tokens by first asking it a question, then giving the model a prompt such that its response evicts the entire KV cache, and then asking the model to recall context from the previous prompts and responses it gave. A full list of the prompts used can be found in our Appendix in Section 7.

Out of the 7 prompts we were able to run our system on, RA-StreamingLLM correctly responded to 3 of them. In addition, on the prompts that RA-StreamingLLM failed to retrieve the correct answer, we saw that when we removed the prompts that caused the entire KV cache to evict, the model still failed to get the right answer for 4 of the prompts. Therefore, some of the failure to respond with the correct answer for the prompt lies within the Vicuña model's ability to respond to prompt instead of the RA-StreamingLLM's ability to retrieve context itself.

4.2 LLM-as-a-Judge

To numerically compare our method against StreamingLLM, we used the LLM-as-a-Judge concept (a method defined in past literature) to evaluate our RA-StreamingLLM against StreamingLLM by utilizing pairwise comparison between the two models and reference-guided grading, where we give the correct answer to the GPT4 and ask it to grade each response [7]. Each prompt group consisted of 3 parts: the initial prompt (which contained information, either in the question or based on the response we would ask later on for the model to retrieve), superfluous prompts that did not contain any relevant information, and finally a question to ask the model about something that was in the first prompt or response. The middle prompt was included to ensure that the original context from the first prompt and answers were evicted.

We evaluated against the same 7 prompts as above, providing GPT4 with the following prompt:

```
You are a grader who is picking between two answers to determine which one
  is better.  Based on the last question in the prompt, answer given, and
```

```
correct answer below, select either answer A or answer B as the better
                                answer.
                            Prompt:  <prompt>
                    The correct answer is:  <answer>
                            Answer A: <answer>
                            Answer B: <answer>
```

We saw that RA-StreamingLLM was determined to have the better answer 5 amount of times, and StreamingLLM was determined to have the better answer 2 amount of times.

Model	Average reference-guided score
RA-StreamingLLM	4.0
StreamingLLM	2.0
LLM with context not evicted (middle prompt removed)	2.0

Table 1: GPT4 LLM-as-a-Judge reference-guided scores

Table 1 shows the average reference guided scores given by GPT-4 for both implementations. For each prompt and asnaswer pair, GPT-4 was given the original prompt itself, the response of the model, and the correct answer to prompt in the following format:

```
   You are a grader who is assigning grades to answers given to a prompt.
Based on the last question in the prompt, answer given, and correct answer
            below, rate the answer given on a scale of 1-10.
                            Prompt:  <prompt>
                    The correct answer is:  <answer>
        Grade the following answer on a scale from 1-10:  <answer>
```

RA-StreamingLLM showed a higher average score than StreamingLLM, indicating its enhanced memory and ability to recall information in comparison to StreamingLLM.

5 Limitations and future work

5.1 Context Window and Generation Size

A limitation of our evaluation and implementation was that we tested this on a smaller context window size of 500 tokens and max generation size of 250 tokens. Testing on larger context windows and generation sizes may impact our results, as the increased amount of embeddings in our vectors DB may lead to a scenario where there are too many similar embeddings or embeddings that may be similar but not necessarily useful as context. This would result our system retrieving the wrong or irrelevant context. In addition, an increased context window and generation size would increase the size of our KV cache database and our vector database, which would thereby increase the processing time needed to query and add into those databases.

Additionally, our approach is still constrained by a hard cap on the context window length. As a result, our approach will fail at tasks that require context from content that is longer than the context window. For example, the model will likely fail at producing a good summary of an entire novel as its contents will likely be too large for models RA-StreamingLLM will be evaluated on.

5.2 Latency and Memory Overhead

Although RA-StreamingLLM does improve context of StreamingLLM, it comes at the cost of added memory overhead due to the fact that the evicted tokens must be stored in a database. In addition, because of this this storage and retrieval, there is added latency to the generation of responses because of the need to retrieve the most similar embeddings from the vector database and also the need to pickle and unpickle the evicted KV cache tokens in the Redis database.

5.3 Language Output

Curiously, the Vicuña model would sometimes return outputs in Chinese, Korean, or other languages when prompted in English. However, the responses it returned were often coherent and correct when answering different prompts. We saw instances of this prior our integration of RAG, indicating that this is likely an issue with the Vicuña model itself and not our added changes to StreamingLLM.

We experimented with different styles and formatting in our prompts to see if we detect any pattern between the specifics of the prompt and the outputted language. So far, we have observed the subtly modifying the prompts affects the output fairly deterministically, but have not been able to generalize to a universal prompt engineering pattern.

5.4 Future work

One area of future work is to develop a more efficient method to utilize RAG in model inference in order to reduce the latency cost discussed previously. A possible way to address this is to change the way we are storing the KV Cache in Redis, as pickling the evicted KV tokens currently causes the most latency in our implementation due to the large size of the evicted KV tokens.

In addition, more extensive testing needs to be conducted to verify these results. Namely, testing with different models larger context window and large max generation size should be done to to ensure that this method is consistent and utilizable across real-world applications of LLMs. This will include testing longer inputs, including testing with larger context window size and max generation size for the LLMs being tested.

6 Conclusion

StreamingLLM is a framework that enables LLMs to handle unlimited texts without fine-tuning, but loses memory due to the tokens it must evict. Through our implementation and testing, we have determined that through our system, RA-StreamingLLM, it is possible to enhance StreamingLLM's memory by using the concept of RAG while maintaining infinite window processing and coherence. Applications of this include utiliziation in long-running conversations, where the model now can remember information "outside the context window" and also being able to retrieve specific bits of information from previous prompts and responses. Our code and demo are publicly available and can be found here.

References

[1] OpenAI. Gpt-4 technical report, 2023.

[2] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[3] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv*, 2023.

[4] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[5] Jerry Liu. LlamaIndex, 11 2022.

[6] Chroma: The ai-native open source embedding database.

[7] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

7 Appendix

7.1 Prompts used to test

A full list of the prompts and their grades by GPT4 can be found here.