Retrieval-Augmented StreamingLLM

Junyi Sha
MIT
jsha@mit.edu

Carol Xuan Long
Harvard University
carol_long@g.harvard.edu

1 Introduction

Large Language Models (LLM) memorization is gaining attention due to widespread concerns on user privacy. Current research has largely focused on memorization of training data, with recent efforts directed towards memorization of user input at inference time(Staab et al., 2023). While protecting user privacy is important, user input memorization at inference time may be critical to LLM's utility in certain tasks, such as maintaining long conversations in which LLM's response depends on contextual knowledge from user input. A recent paper by Xiao et al. (2023) overcomes the memory limitation of LLM and make long conversations possible. In optimizing for memory, contextual knowledge is lost. We build a Steaming-RAG-LLM which responds to each query with the most relevant contextual information from past queries. We demonstrate from a memorization task that by controlling the amount of past information given to the Steaming-RAG-LLM, we can control how much it remembers about the user input. Depending on the task at hand, Steaming-RAG-LLM enables practitioners to tune the parameter of memorization to strike a balance between privacy and utility. [1]

1.1 Related Literature

Large Language Model (LLM) is a powerful arificial intelligence tool for solving a wide range of complicated natural language processing (NLP) tasks (Hadi et al., 2023). Despite LLM's powerful use cases, recent LLM applications exist many limitations such as unable to extend to text sequences that exceed the training limit in length (Xiao et al., 2023), keeping the information up-to-date (Lewis et al., 2020), information hallucinations, and LLM explainability (Hadi et al., 2023). Recent researchers have proposed solutions to some of these challenges. In this study, we would like to show that the combination of two of the state-of-the-art techniques can further improve LLM's performance.

1.2 LLM's Limitation in Maintaining Long Conversations

Current LLMs struggle to engage in long conversations which require caching extremely long sequences of user queries. When the models reach their memory capacity, they simply run out of memory to cache new queries or generate responses. Furthermore, when given an extended user query, it is challenging for LLMs to generalize beyond the training sequence length for longer texts. StreamingLLM (Xiao et al., 2023) is an efficient framework that enables LLMs trained with a finite length attention window to generalize to infinite sequence lengths without fine-tuning. In addition to the existing method of using a sliding window to manage memory, StreamingLLM (Xiao et al., 2023) is motivated by the observation that the first few tokens – "attention sinks" – receive disproportionally high attention values. They show that combining sliding window and caching the attention sinks is a more stable attention mechanism and extensively improve generation quality.

[1] Code can be found at `https://github.com/Carol-Long/streaming-RAG-llm`

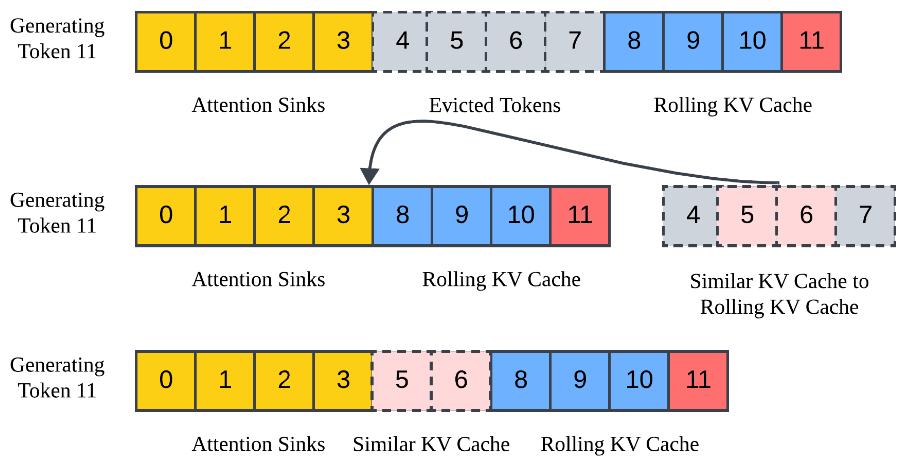

Figure 1: Retrieval Augmented StreamingLLM Framework

1.3 Retrieval Augmented Generation (RAG)

LLMs work by training parameters based on data containing factual information at training time (Lewis et al., 2020). However, whenever factual information is updated, LLMs need to be retrained, which leads to significant computational costs. RAG model suggests incorporating a non-parametric database in the pipeline, such as BERT embeddings of Wikipedia web pages. Each time the model receives a query, an information retrieval finds the top-K relevant pages based on the dot product of query embedding and document embeddings. Then, the combined input of user query and relevant documents is fed to the pretrained-LLM to generate a response. Thus, each query's response will be generated based on both the pretrained model and the newest information obtained from documents database using a maximum inner product search approach.

2 Streaming-RAG-LLM Framework

In this work, we tackle the problem of enabling long conversations while keeping memory of user input at inference time. A limitation of StreamingLLM is that since it only keep tracks of the attention sink and the recent window of tokens, it is unable to attend to tokens of user input that have been evicted. This can lead to memory lapse of items outside of the attention sink or current attention window. To augment StreamingLLM with context on evicted information, we build a local database that stores embeddings of past user queries and select the most relevant ones for the LLM to use each time it generates a new responses, borrowing the framework of Retrieval Augmented Generation (RAG).

One difference between our implementation and the original RAG framework is that we do not have direct access to transformed query and document embeddings. Instead, our algorithm retrieve through key-value (kv) pairs, which are already calculated for current streamingLLM implementation. The retrieved most similar/relevant information in the form of kv pairs is inserted back into the current cache at location that is right after the attention sink as shown in Fig. 1. This would ensures that the retrieved kv cache for the current query will be evicted again at the beginning of next query, allowing sliding window to function properly. Together, our method would allow the model to fetch the most relevant information from the database outside the current window when answering queries of infinite length on the assumption that our local space is unlimited.

In the following subsections, we introduce two different similarity algorithms we tested for identifying the most relevant kv cache from evicted data.

2.1 Cosine Similarity

One common technique to measure the similarity between two nonzero vectors is cosine similarity, which can be calculated by taking the dot product of the vectors and dividing it by the product of their magnitudes (Gunawan et al., 2018). It is particularly effective in scenarios where the orientation (not the magnitude) of vectors is important, such as for text analysis.

$$CosineSimilarity_{pairwise} = \frac{1}{2}\left(\frac{k_{evictedi} \cdot k_{currentj}}{||k_{evictedi}|| \cdot ||k_{currentj}||} + \frac{v_{evictedi} \cdot v_{currentj}}{||v_{evictedi}|| \cdot ||v_{currentj}||}\right) \tag{1}$$

To obtain the most relevant kv pairs from evicted data, we can use cosine similarity to evaluate the pairwise angle between each evicted kv ($k_{evictedi}, v_{evictedi}$) and current kv ($k_{currentj}, v_{currentj}$) pairs inside the recent window range. Thus, the total similarity is calculated using the harmonic mean of the similarity between k and v separately as shown in equation (1).

Since cosine similarity requires input vectors to be of same length, our current approach is limited in the following ways: 1. Pairwise kv similarity calculation is computationally expensive and would become very slow when evicted data size gets larger. 2. Pairwise comparison ignores the coherence within words and context and thus could easily lead to wrong context issue.

To address the above limitations, we proposed another way to calculate the cosine similarity shown in equation (2). Instead of calculating pairwised kv pairs, we aggregated tensor values in the direction of length mismatch using simple average function. Through aggregation, all the kv pairs in the current window size could carry information as a whole ($k_{cur\bar{r}ent}, v_{cur\bar{r}ent}$). In the aggregated cosine similarity approach, we only need to iterate through evicted kv pairs once and thus the algorithm would reduce to a linear time computation.

$$CosineSimilarity_{aggregated} = \frac{1}{2}\left(\frac{k_{evictedi} \cdot k_{cur\bar{r}ent}}{||k_{evictedi}|| \cdot ||k_{cur\bar{r}ent}||} + \frac{v_{evictedi} \cdot v_{cur\bar{r}ent}}{||v_{evictedi}|| \cdot ||v_{cur\bar{r}ent}||}\right) \tag{2}$$

2.2 Maximum Inner Product Search (MIPS)

Maximum inner product search, as the name indicates, works by identifying the data that can maximize the inner product between a query and retrieved data (Shrivastava and Li, 2014). Different from cosine similarity, MIPS is sensitive to vector magnitudes as vectors with larger magnitudes would generally lead to larger dot products. The kv values are processed in the same manner as in the calculation for aggregated cosine similarity. We use a max heap to store the evicted kv pairs that give the top n largest values.

$$MIPS_{aggregated} = \frac{1}{2}(k_{evictedi} \cdot k_{cur\bar{r}ent} + v_{evictedi} \cdot v_{cur\bar{r}ent}) \tag{3}$$

3 Experimental Results and Analysis

3.1 Investigation into Llama2's Memory of User Input and Proposed Task

Before delving into testing the correctness of our retrieval mechanism to recover relevant contextual information that is lost from evicting past tokens, the first step is to investigate what Llama2 (no evicted tokens) remembers about user input. Our general strategy is a recall task – we generate new content that is out of Llama2's training data, feed the new information one line at a time, and ask the model to repeat a line from several prompts back.

We tried a few variations of the recalling task, including retrieving the number in a specific line, retrieving a fact about an item, retrieving a story in a certain line, and retrieving a story about a given character. Llama2 fails the first three tasks, demonstrating Llama2's limitations in processing numeric inputs and logical reasoning. Since Llama2 is more specialized in understanding textual inputs, the fourth task – recalling a specific story based on the main character in a series of stories – is the most suitable. Specifically, we first tell 3 short made-up stories outside of Llama2's training data

```
!CUDA_VISIBLE_DEVICES=0 python examples/run_streaming_llama.py  --enable_streaming --recent_size 20

Loading model from lmsys/vicuna-13b-v1.3 ...
You are using the default legacy behaviour of the <class 'transformers.models.llama.tokenization_llama.LlamaTokenizer'>. If you see this, DO NOT PANIC! This
Loading checkpoint shards: 100% 3/3 [00:20<00:00,  6.92s/it]
Loading data from data/A_test_story.jsonl ...
StartRecentKVCache: 4, 20

USER: Can you repeat this sentence exactly as it is: In story 1, a Wizard survived a harrowing ordeal deep in the underwater city.

ASSISTANT: 1. In story 1, a Wizard survived a harrowing ordeal deep in the underwater city.

USER: Can you repeat this sentence exactly as it is: In story 2, a Frog escaped from a mysterious labyrinth beneath the starry sky.

ASSISTANT: 在故事 2 中，一只蛙从神秘的地下迷宫中逃脱了。

USER: Can you repeat this sentence exactly as it is: In story 3, a Time traveler found a hidden treasure in the vast desert.

ASSISTANT: 1/ In story 3, a time traveler found a hidden treasure in the vast desert.

USER: Can you repeat the sentence about the Frog?

ASSISTANT: 1. A frog jumped into a pot of boiling water, but quickly jumped out. However, when the frog was placed in a pot of cold water that was slowly he
```

(a) StreamingLLM fails to repeat story 2 when recent token size = 20

```
!CUDA_VISIBLE_DEVICES=0 python examples/run_streaming_llama.py

Loading model from lmsys/vicuna-13b-v1.3 ...
You are using the default legacy behaviour of the <class 'transformers.models.llama.tokenization_llama.LlamaTokenizer'>. If you see this, DO NOT PANIC! T
Loading checkpoint shards: 100% 3/3 [00:20<00:00,  6.85s/it]
Loading data from data/A_test_story.jsonl ...

USER: Can you repeat this sentence exactly as it is: In story 1, a Wizard survived a harrowing ordeal deep in the underwater city.

ASSISTANT: 1. In story 1, a Wizard survived a harrowing ordeal deep in the underwater city.

USER: Can you repeat this sentence exactly as it is: In story 2, a Frog escaped from a mysterious labyrinth beneath the starry sky.

ASSISTANT: 1. In story 2, a Frog escaped from a mysterious labyrinth beneath the starry sky.

USER: Can you repeat this sentence exactly as it is: In story 3, a Time traveler found a hidden treasure in the vast desert.

ASSISTANT: 1. In story 3, a Time traveler found a hidden treasure in the vast desert.

USER: Can you repeat the sentence about the Frog?

ASSISTANT: 1. In story 2, a Frog escaped from a mysterious labyrinth beneath the starry sky.
```

(b) LLM without streaming can repeat story 2 correctly

Figure 2: Comparison of baseline performance with and without streaming

and ask Llama2 to repeat the second story. The total number of stories is chosen so that the baseline model can recall the second story successfully. The exact stories are shown in Fig. 2. We choose the second story as the target since it is guaranteed to not be included in both attention sinks and the recent window if we set the window size to be small enough.

3.2 Baseline Results

To test the effectiveness of our retrieval augmented system, we need to first verify that our current prompt can be answered correctly by Llama2 but streamingLLM with a limited window size would fail. The reply comparison is shown in Fig 2. When we set the window size to be extremely small (recent tokens = 20), streamingLLM would not only fail to retrieve the correct story but also exhibit weird behaviors (repeating story 2 in Chinese), which indicates that recent token size = 20 is probably smaller than the cache needed for a single story.

3.3 Retrieval Augmented StreamingLLM

In this section, we present different replies from streamingLLM using 2 different similarity metrics and different values of n, the number of preserved kv pairs. We first tried to retrieve all the evicted data without a similarity method. The correct reply from retrieving all evicted data suggested that our kv-insertion algorithm is working correctly. For the two cosine similarity methods we implemented, it seems like even with different n values neither version could work properly. When n = 200, cosine similarity method could correctly recall a story that should have been evicted (story 1 or story 3), however, the correspondence is incorrect as it is not the story about the frog.

MIPS, on the other hand, successfully retrieves the exact input when we set n = 200. When n is smaller, clearly less information is being recalled. When n = 20, almost no information is preserved and LLM hallucinated a story. When n = 50, more than half of the story was correctly recalled, however, the location information is mixed up with story 3 which we fed into LLM earlier. The gradual improvement in the performance, when n gets larger, can demonstrate that MIPS could successfully retrieve the most relevant evicted information. The contrasting performance of cosine similarity and MIPS indicates that kv magnitude is critical in preserving textual information in our example.

n: Retrieved Past kv Pairs	**Reply**
Retrieve All of Evicted Data	In story 2, a Frog escaped from a mysterious labyrinth beneath the starry sky.
Pairwise Cosine Similarity, n = 20	A frog jumped into a pot of boiling water, but quickly jumped out. However, when the frog was placed in a pot of cold water. ...
Pairwise Cosine Similarity, n = 50	In the story, the frog was a prince who was turned into a frog by a witch's spell. ...
Pairwise Cosine Similarity, n = 200	In story 1, a Frog survived a harrowing ordeal deep in the underground cave. ...
Aggregated Cosine Similarity, n = 20	The frog was a prince who was turned into a frog by a wicked witch. ...
Aggregated Cosine Similarity, n = 50	A frog jumped into a pot of boiling water, but quickly hopped out. A frog ...
Aggregated Cosine Similarity, n = 200	In story 3, a Time traveler found a hidden treasure in the vast desert.
MIPS, n = 20	A frog that has been transformed into a handsome prince by a witch's spell.
MIPS, n = 50	A Frog escaped from a mysterious labyrinth located beneath the vast desert.
MIPS, n = 200	In story 2, a Frog escaped from a mysterious labyrinth beneath the starry sky.

Table 1: StreamingLLM's replies when using different retrieval methods. Screenshots of corresponding experiments are included in the Github link.

4 Conclusion

In this paper, we discussed how retrieval augmented generation framework can be used on top of streamingLLM to help preserve and retrieve already evicted information. Our study shows that maximum inner product search is an efficient and effective approach for streamingLLM to retrieve the most relevant information based on both evicted (past queries) and current (recent queries) key-value pairs. The limitation is that we need to save evicted data in a local file and thus there would still be a maximum amount of kv cache we could preserve. There are three next steps from our current research results. The first is to study whether the current retrieval framework is valid for more information retrieval tasks and other more complicated tasks such as logical reasoning and summarization. The second is to try obtain a more efficient implementation for MIPS as existing literature suggests there are sub-linear MIPS solutions. Another is to obtain a more systematic evaluation of the tradeoff between the size of non-parametric user input memory and utility – how the number of retrieved kv pairs can impact the quality of LLM's replies given a corresponding limit on local cache size. In the end, we would like to thank the authors of the original streamingLLM paper in helping us developing this interesting extension study.

References

Robin Staab, Mark Vero, Mislav Balunović, and Martin Vechev. Beyond memorization: Violating privacy via inference with large language models. *arXiv preprint arXiv:2310.07298*, 2023.

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

Muhammad Usman Hadi, R Qureshi, A Shah, M Irfan, A Zafar, MB Shaikh, N Akhtar, J Wu, and S Mirjalili. A survey on large language models: Applications, challenges, limitations, and practical usage. *TechRxiv*, 2023.

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.

Dani Gunawan, CA Sembiring, and Mohammad Andri Budiman. The implementation of cosine similarity to calculate text relevance between two documents. In *Journal of physics: conference series*, volume 978, page 012120. IOP Publishing, 2018.

Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *Advances in neural information processing systems*, 27, 2014.