Retrieval Augmented Generation and Attention Sink Capitalization in Streaming Language Models

Tony Cui, Kevin Tong, Anish Ravichandran
Massachusetts Institute of Technology
tacui@mit.edu, kevinct@mit.edu, anishrav@mit.edu

Abstract

Though transformers have transformed the field of Natural language processing, it has always been restricted to limited context windows or expensive computation. With the Introduction of StreamingLLMs, transformers can now seemingly generate new tokens indefinitely and efficiently. In this paper, we introduce the RememberingLLM, a build-on to the StreamingLLM, which utilizes Retrival Augmented Generation to add back lost information back into the KV Cache, and uses K-clustering and the silhouette method to capitalize on the attention sink to hold retain primary ideas of the original context.

1 Introduction

Transformers have been revolutionary in the field of Natural Language Processing tasks, particularly in summarization, question and answering, and text generation.

Though transformers have been proven effective in many NLP tasks, they have always bounded by the limitations of complexity and context length. Particularly, once a transformer generates tokens beyond its trained context length, its perplexity raises to high levels. Through using attention sinks and implementing on a KV Cache, streaming language models can now generate tokens seemingly indefinitely using a rolling context window with attention sink, and generation complexity was reduced from $O(T^2)$ to $O(TL)$ where $T >> L$. However, it cannot generate tokens while pertaining to previously evicted tokens. Xiao et al. [2023]

We introduce Remembering LLMs, which uses two methods to fix this problem. First, we store evicted tokens in a vector database and use Retrieval Augmented Generation (RAG) to add back evicted tokens into the KV Cache. In addition, we introduce the idea to capitalize on the attention sink. With RAG, we hope to improve metrics on long Q&A, summarization, and story generation tasks.

In this paper, will share in detail our methodologies, experiments, and conclusions.

2 Related Works

In this section, we summarize related works pertaining to the modifications we make to solve new issues with StreamingLLM

2.1 StreamingLLM

StreamingLLM is used for deploying Large Language Models (LLM) in streaming applications, such as multi-round dialogues. While ideally one would cache all of the previous text seen thus far (in the form of tokens and key-value pairs), this would lead to a large amount of memory usage. Furthermore,

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

many Popular LLM models can only be applied to text up to a certain training sequence length. As a result, it becomes necessary to sacrifice some of the accuracy of the LLM for performance.

StreamingLLM combines two ideas to solve these problems. Firstly, StreamingLLM utilizes a sliding window approach to keep track of the most recent tokens and key value pairs seen in the dialogue so that the LLM can use that data to develop its response. This alone would give the LLM the most recent context needed for developing its answer. However, it begins to fail when the length of the dialogue exceeds the cache size, causing the LLM to forget the previous context.

StreamingLLM also deploys the use of an attention sink, which provides the earliest couple tokens and key value pairs in the dialogue, which turns out to provide enough context for the LLM to have most efficiency and stability up to four million or more tokens. Furthermore, because the LLM only uses cached key-value pairs and tokens from the sliding window and attention sink, it also exhibits a 22.2 times speed up over the sliding window recomputation baseline model [1].

One limitation of the StreamingLLM is when working with large texts summaries or long dialogues, the StreamingLLM may evict the relevant tokens and key-value pairs from its cache, causing it to not have the information it needs to produce the correct response. In the next section, we introduce our RememberingLLM, which builds on top of StreamingLLM but utilizes a vector store, k-clustering, and modifications to the KV Cache in order to fix this problem.

2.2 Retrieval Augmented Generation

Retrieval Augmented Generation Lewis et al. [2021] was a technique introduced in 2020 to help language models generate tokens based on ground truth stored in a database. A typical use case is having documentation be chunked and transformed into lower dimensional vector encoding using embedding models. When a query is asked, a similarity metric, such as cosine similarity, is applied to find the k most relevant chunks of information. The information can then be appended to language model prompts and generate answers grounded on truth.

Many popular vector databases uses the HSNW algorithm Malkov and Yashunin [2018] perform the most expensive task in this process apart from token generation, the top k query, in average $O(\log n)$ complexity.

2.3 K-Means Clustering

K-means clustering is a classic unsupervised learning algorithm used to label n datapoints to k vectors while minimizing within-cluster variances. We have not yet found instances where researchers have attempted to use K-means clustering to add sparse coverage of context for generative models.

3 RememberingLLM

In this section, we provide a breakdown of the different components of RemenberingLLM, and how it improves on top of the original StreamingLLM.

3.1 Expanding the KV Cache for Contexts

In order to support the RememberingLLM, we must make changes to the KV Cache so that it can hold not only the attention sink and the sliding window of tokens, but also the additional context that we add to the model so that it “remembers" relevant information.

To do this, we create an additional section to the KV Cache between the attention sink and the sliding window section. This additional section starts off with size 0 since there is no additional context at the beginning that needs to be remembered. Instead, we initialize this space in the KV Cache after the first eviction of tokens from the KV Cache.

In addition to creating the context section in the KV Cache, we also must store all the tokens and key value pairs we’ve seen so far so that we can use it for identifying relevant information later on. The KV Cache will now store an index indicating the next key-value index that must be saved for the context. We then save all new key-values on and after this index whenever new key-values are added.

At the same time, we save all the individual tokens as well as we come across them, either from the input (the prompt), or from the text generated by the large language model.

This way, we have a one to one mapping from the tokens to the key value pairs by looking at the index within each of these two lists. We can then use the k-clustering and vector store to obtain the relevant indices from these lists and input them into our key-value cache before passing them onto the model.

3.2 Retrieval Augmented Generation on Evicted Tokens

For retrieval augmented generation, we are using ChromaDB as a vector database for vector store and querying. We decide to keep the vectors as 100 consecutive tokens. That is, whenever we get 100 more tokens, we combine them into text and add them into the vector store. In addition to the text, we also store the index range these texts represent as metadata and use this to extract the relevant key-value pairs later when adding in the context.

Each time we encounter a new prompt, we query the vector database for the 3 most relevant texts and add them into the context section of the KV Cache so that the model and "remember" the context it needs to answer the prompt well. Since we get the 3 most relevant texts, and each text has 100 tokens in it, we decide to set the context size to be 300 for the KV Cache.

Through these changes, we expect that the model will be able to answer prompts accurately, even if the relevant information has already been evicted. Since we get the three most relevant pieces of text from the vector store, we hope that the resulting response from the model is able to utilize enough relevant information to be satisfactory.

3.3 Attention Sink Capitalization

Currently, streaming models only attend to the first four tokens of a context. It has been shown that you only need about four tokens to attend to at the start to maintain low perplexity, and even if they were '\n' characters Xiao et al. [2023]. We believe that we could achieve greater results if we could increase the size of the attention sink, and have relevant information within it.

To choose what goes inside the new attention sink, we use the K-means clustering algorithm to create k separate clusters of chunks from the original input, using the silhouette method to obtain a heuristic on what the best value of k would be. We select the k such that the silhouette metric would be the greatest. We can then sample small chunks from each of the k clusters, and place them in our new attention sink. This would especially be promising for multi-document tasks such as multi-document Q&A and Summarization, where there will be natural boundaries for each cluster.

4 Experiments

For the experiments, there are a few tasks we are aiming to improve upon: Long Question and Answering, Multi-document Summarization, and Story Generation.

For Multi-Document Summarization and Answering, the current top model on the Multi-News dataset is PRIMER with a ROGUE-1 score of 49.9, using a Pyramid-based Masks. Xiao et al. [2022]

For Story Generation, the top model for the Writing Prompts dataset is the BART model, having a BLEU score of 33.79.

4.1 StreamingLLM + Retrieval Augmented Generation

By the time we integrated the RAG methodology and expanded KV Cache into RememberingLLM, we had realized that we did not have enough compute to properly evaluate the implementation. Just running the baseline perplexity metric on the StreamingLLM with an expanded KV cache took over 10 minutes for a 194 token sample. Unfortunately, we cannot give complete results at this time, though we can provide a demo.

4.2 StreamingLLM + Attention Sink Capitalization on K-clustered samples

Though we were able to implement K-means clustering and choosing optimal k through silhouette on long documents, we ran out of time to implement this methodology into RememberingLLM. We believe this strongly method has potential to improve results on NLP tasks.

4.3 StreamingLLM + Combined Methods

Given the state of the two previous experiments, we are not able to provide full metrics for the method combining RAG and K-clustered samples in the attention sink. We hope to explore this method in future research.

5 Conclusion

Though RememberingLLM was not able to be properly evaluated, the theory behind taking advantage of the attention sink with RAG and using clustering to retrieve the most important information holds promise. Currently, the current attention sink adds no relevant context for streaming models to attend to. In addition, Retrieval Augmented Generation has not been added to streaming models yet.

Given more time to complete this project, the team is confident promising results would have been delivered. Future avenues of research include performing the experiments with RAG and K-clustered sampling for attention sink capitalization and their combination.

6 Future Research

For updates on further research on RememberingLLM, please visit our Github work at https://github.com/anish219/remembering-llm

References

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

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

Yu. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, 2018.

Wen Xiao, Iz Beltagy, Giuseppe Carenini, and Arman Cohan. PRIMERA: Pyramid-based masked sentence pre-training for multi-document summarization. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5245–5263, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.360. URL `https://aclanthology.org/2022.acl-long.360`.