

# An Experiment of Implementing Reservoir Sampling to StreamingLLM

Jiyang Pan<sup>1\*+</sup>,

Jiayi Weng<sup>2+</sup>,

Yansen Huang<sup>3+</sup>,

Patrick Pan<sup>4+</sup>,

Qianbiao Zhao<sup>5+</sup>

<sup>1</sup>Department of Information systems and management, Najing Audit University, Wuxi, China, 201743319@qq.com,

<sup>2</sup>Fazekas Mihaly Secondary School of Budapest, Budapest, Hungary, jiayiweng10@gmail.com,

<sup>3</sup>Guangdong Experimental High School International Department, Guangzhou, China, jasonhuang2007@139.com,

<sup>4</sup>Suzhou Fenghua School, Suzhou, China, patrickpym@163.com,

<sup>5</sup>Guangdong Country Garden School, Foshan, China, qianbiaozhao92@gmail.com

## Abstract:

When the text is longer than the training sequence length, the streamingLLM can successfully improve the computational speed and guarantee a certain degree of accuracy. But this method is only suitable for short term memory questions and answers. Because the StreamingLLM doesn't improve a computer's Long-term memory ability. We tried to combine the reservoir sampling with streamingLLM. Since StreamingLLM, the reservoir sampling will randomly take samples from what were meant to be discarded. Through adding reservoir sampling, we find the results are more accurate and representative.

**Keywords:** streamingLLM, Reservoir Sampling, Large Language Models, KV-cache, tokens

## 1. Introduction

We introduce an attention mechanism that is based on the StreamingLLM framework. Taking the use of its code we melted into an idea of another attention mechanism called HyperAttention [1]. The foundational principle of StreamingLLM is the concept of an attention sink. This means that by preserving the KV pairs of the first tokens, the performance of the

attention mechanism largely recovers. This enhancement occurs because of the high attention score these tokens get, although they are not as important as we would think of them. As a result, the StreamingLLM has demonstrated up to 22 times speedup compared to the sliding window method.

Our research team sought to further enhance this streaming processing technique, in terms of either boosting its accuracy or achieving additional speed-

up. We explored the potential of integrating concepts from HyperAttention, a method described in a recent paper that achieved a linear-time sampling algorithm even with unbounded matrix entries. A critical component of their approach is causal masking. To specify this, authors of HyperAttention were looking for a sampling matrix with a limited number of rows to address the computational challenges. The mentioned sampling strategy was what we combined with the notion of attention sink from StreamingLLM. Instead of immediately deleting the heavy entries in the KV-cache, as done in StreamingLLM, we implemented a function called the reservoir sampling into the original code. As a result of this adjustment, a sample matrix that has a fixed size will be created. In addition, as new tokens arrive in the stream, the algorithm decides whether to include them in the sample matrix. Initially, for the first  $k$  items, they are directly added to the sample. However, for subsequent items, the reservoir sampling process ensures each item has an equal probability of being included. The primary objective of integrating the reservoir sampling method is to manage memory and space usage. By applying this method, we aim to achieve constant usage of those. Moreover, processing a smaller yet representative sample matrix can substantially reduce computational requirements, potentially leading to a more

efficient streaming model.

## 2. Background

StreamingLLM is a simple and efficient framework, which can deal with text of infinite length without fine-tuning. StreamingLLM trains the large language models with a finite attention window [2]. It keeps the attention sink tokens' KV (the four initial tokens) together with the sliding window's KV to anchor the attention computation and stabilize the model's performance [3]. But why are the four initial tokens' KV 'so essential? The researchers found that the model consistently focuses on the initial tokens. It means that deleting these initial tokens' KV will remove a large portion of the denominator in the SoftMax function in the attention calculation. It is evident that the puzzle increases when the text length exceeds the cache size, led by the exclusion of initial tokens.

## 3. Code

To improve the efficiency and maintain the stability of the LLMs [4], the streaming large language models set up a limited attention window to train the LLMs. Because the memory space is limited, the code only reserves the four initial tokens (the `start_size` in code) and the recent tokens.

```
torch.cat(
    [
        self.k_slice(k, 0, self.start_size),
        self.k_slice(
            k, seq_len - self.recent_size + num_coming, seq_len
        ),
    ],
    dim=self.k_seq_dim,
),
torch.cat(
    [
        self.v_slice(v, 0, self.start_size),
        self.v_slice(
            v, seq_len - self.recent_size + num_coming, seq_len
        ),
    ],
    dim=self.v_seq_dim,
```

**Figure1. The detailed code about initial tokens and recent codes.**

As shown in Figure1, if the current sequence length plus the number of new elements is greater than the cache size (`self.cache_size`), the code will delete some elements and make room for the new tokens. The purpose of this

method is to ensure that the cache does not exceed a predetermined size, while preserving as many important tokens' KV as possible (the start tokens and the recent tokens). This is an effective strategy for managing memory

when working with long sequences of data, especially in decoding or generation tasks where the model needs to access recent context information.

```
def reservoir_sampling(self, seq_len):
    indices = list(range(self.start_size, seq_len - self.recent_size))
    sample = []
    for i in range(len(indices)):
        if i < self.sample_size:
            sample.append(indices[i])
        else:
            j = random.randint(0, i)
            if j < self.sample_size:
                sample[j] = indices[i]
    return sorted(sample)
```

**Figure 2. The reservoir\_sampling code.**

However, we changed this method of removing intermediate data indiscriminately. As shown in figure2, we select some tokens from what were supposed to be deleted by the reservoir\_sampling algorithm[5], which is used to draw a fixed number of samples uniformly and randomly from a larger dataset without the need to store the entire dataset. When the sample size(self.sample\_size) is less

than the number of candidate indexes, the code will reserve the initial tokens and the recent tokens firstly. Then, it will randomly select  $n(n = \text{sample\_size} - \text{start\_size} - \text{recent\_size})$  tokens from the remaining sequence. As a result, the attention sink isn't just the initial four tokens' KV and the recent tokens' KV. It also includes the extra random sample in the middle sequence.

```
sample_indices = self.reservoir_sampling(seq_len + num_coming)
sample_slices_k = [self.k_slice(past_key_values[0][0], i, i+1) for i in sample_indices]
sample_slices_v = [self.v_slice(past_key_values[0][1], i, i+1) for i in sample_indices]

torch.cat(
    [
        self.k_slice(k, 0, self.start_size),
        *sample_slices_k,
        self.k_slice(k, seq_len + num_coming - self.recent_size, seq_len + num_coming),
    ],
    dim=self.k_seq_dim,
),
torch.cat(
    [
        self.v_slice(v, 0, self.start_size),
        *sample_slices_v,
        self.v_slice(v, seq_len + num_coming - self.recent_size, seq_len + num_coming),
    ],
    dim=self.v_seq_dim,
```

**Figure 3. The combine between start\_slices, sample\_slices and recent slices.**

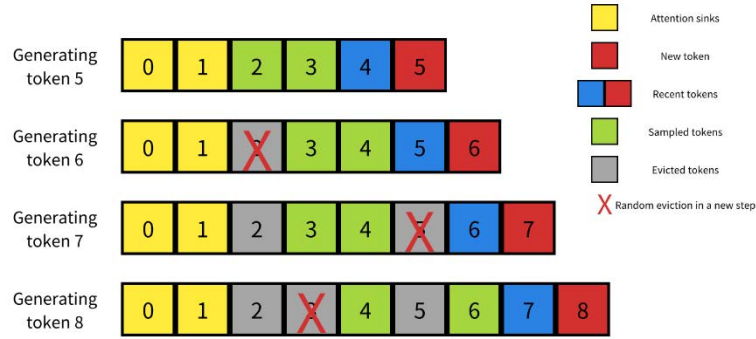
As shown in figure3, we use the sample\_indices to generate the sample slices. Next, we combine the start\_slices, sample\_slices and recent slices by the torch.cat. The key to this method is that each time a new token is added, the code will carry out a reservoir\_sampling algorithm [6], instead of simply discarding all the in-between tokens. This helps reduce memory usage when working with long sequences of data, while preserving the information that is most important to the task at hand.

## 4. Methodology of reservoir sampling implementation

Through reading and discussion, we decided that our basic idea is to combine the ideas of HyperAttention into StreamingLLMs and our codes are based on the codes of the paper Efficient Streaming Language Models with Attention Sinks as the basic codes [7, 8]. To be more specific, to keep the kv-cache at a limited size, the original codes

keep the first few starting tokens and most recent tokens whereas the middle ones will be evicted. So instead of

evicting all the tokens in the middle, we add a space for storing a random sample of the middle ones.



**Figure 4. Example of KV cache changing.**

In this simple example shown in Figure 4, `start_size`, `sample_size` and `recent_size` is both set to 2. By adding up the three sizes, the limitation of the size of the kv-cache is 6. When the size of the cache is smaller than the limitation size, all tokens will be kept. When the cache size gets over the limitation 6, two of the “middle part”, will be randomly sampled (not common completely random sampling, will be explained in the next paragraph), and tokens that are not sampled will be evicted. So the calculation will be based on the 6 ones that are kept, two attention sinks, two sampled tokens, and two recent tokens. Since we cannot get the evicted tokens back and the total size of the stream is unknown, we use reservoir sampling to make sure that every token has the same probability to be sampled. To do the reservoir sampling [9], we introduce another variable `total_num`(total number of tokens that have come in). Starting from the kv-cache reaching the limitation size, each token, which leaves the “recent tokens” part and becomes a new member of the “middle part”, has a probability  $p$  to evict and replace a random old sample, and a probability  $1-p$  to evict itself. In the situation that it replaces an old sample, each old sample has an equal probability  $q$  to be replaced.

$$p = \frac{\text{sample\_size}}{\text{total\_num} - \text{start\_size} - \text{recent\_size}} \dots (1)$$

$$q = \frac{1}{\text{sample\_size}} \dots (2)$$

For example, in the step “generating token 6” in Figure 4, token 4 leaves the “recent tokens”, so it gets a  $p=2/(7-2-2)=2/3$  to be kept. Here we choose to let it be kept, so both token 2 and token 3 have a probability 1/2 to be replaced by token 4. Here we choose token 2 to be replaced by token 4. In the next step “generating token 7”, token 5 leaves the “recent tokens”, so it gets a  $p=2/(8-2-2)=1/2$  to be kept. Here we choose the opposite situation which has probability  $1-p=1-1/2=1/2$  that it evicts itself, so token 3

and token 4 are kept. In the next step “generating token 8”, token 6 is kept with the probability  $p=2/5$ , so both token 3 and token 4 have a probability 1/2 to be replaced by token 6 and here token 3 is replaced. As new tokens are coming in, this process will be repeated and the size of the kv-cache will always be the same.

## 5. Problems we met and relevant solutions

During our scientific research, researchers need to complete the task according to the code of the algorithm in the paper: First, researchers need to reproduce all the code in the paper and run it successfully. After understanding the code, researchers need to rewrite the code to add the function of Hyper Attention in streaming-LLM [10]. However, throughout the time of reading and writing the code, the team encountered various problems. There are 5 main problems. The first main problem was that when researchers tried to reproduce the code in pycharm, researchers could not find a package called triton. This package improves the speed of model reasoning and training by providing a high-level programming language and compilation tool chain that enables developers to write efficient GPU cores. When researchers looked up the reason on Google, researchers found that triton only supports linux operating systems, not windows. Then researchers downloaded the compiled package and installed it to run locally, but python still reported some compilation errors that could not be corrected. The second major issue concerns some file import issues on Google Colab. The authors of the code have written many separate code files to assist in running hyper-attention or to implement some of the unique methods hyper-attention has. Therefore, when running the code file that implements hyper-attention, researchers need to import all the files related to the code, otherwise hyper-attention will not run. But when research-

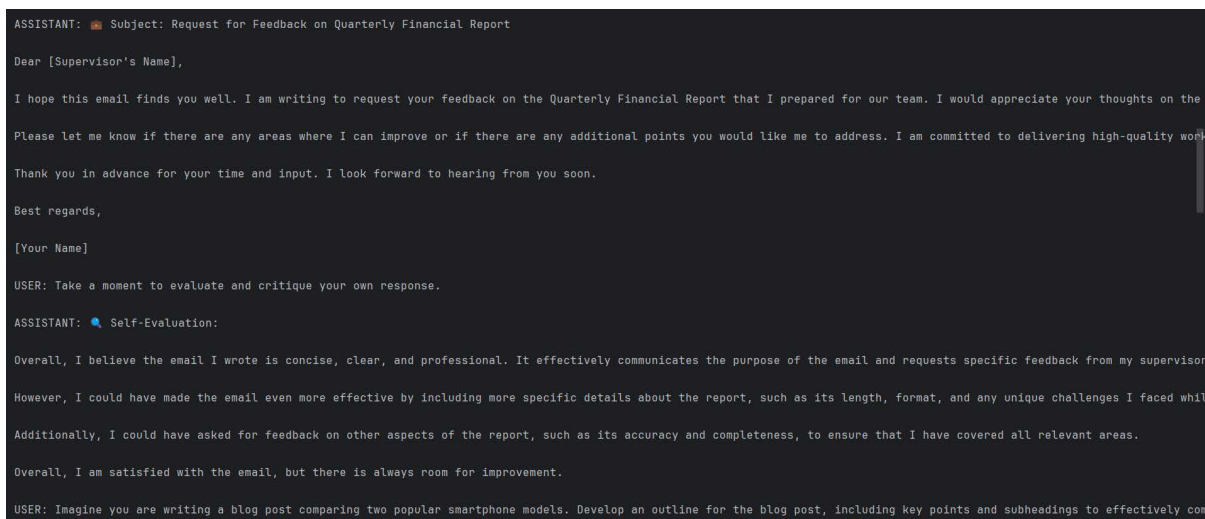
ers run this code in Google Colab, errors always occur. The third main problem is the version problem of some packages. Some of the third party packages that were downloaded were too old or too new to match our code, making the overall error not work. There are also some third-party package versions that are not available online. The fourth major problem is SSL errors. When researchers need to connect to some model site to download the pre-trained model to implement the code, researchers need to turn on the VPN, otherwise researchers cannot do anything on Google. The main reason is that firewalls in some countries or regions block VPN traffic, which can cause SSL connections to fail. The firewall may detect VPN traffic and actively block it, making it impossible to establish a secure SSL connection. At this point, researchers were in a dilemma: whether to turn on the VPN or not? The last major issue is about the model itself. Some models contain so much data that downloading them can take a lot of time. There are also models whose types do not match the functionality of our code, and there are even models that are not open to the public.

There are some relevant solutions that can solve these problems. For the first question, researchers found that codes can run on Google Colab because Colab comes with several third-party packages necessary for the machine learning space, including triton. For the second question, researchers saved the unique file code to Google Cloud Drive first, so the path of the file code is known. Then researchers imported the os package and wrote some methods to modify the current path of the file code so that

python could find and use the file code to run hyper-attention properly. As for the third question, our team invited a teaching assistant to help us solve the problem: download some of the larger version related and compatible packages, and those packages generally have the same effect and functionality. For the fourth problem researchers wrote a string of code to solve it: researchers set the URL and port of the HTTP and HTTPs proxy servers. In this way, we can download public models on some websites without VPN. For the last question, we found that there are still many models available on the website, and what we can do is not to rewrite these existing mature models, because it is not practical and difficult, so we choose to try one by one until we find a suitable model. Fortunately, there is a suitable one and the code reported no errors and returned a respectable result.

## 6. Results and discussion:

After overcoming the aforementioned obstacles, we successfully ran the new program on a local PC instead of Google Colab. The primary issue with Google Colab was that the server would crash once the memory was used up. This problem was mitigated by using a local PC. Initially, the model ran on the CPU, which proved inefficient. Eventually, as shown in figure5, we add some parameters and additional codes to utilize the GPU, this significantly reduces training times and enables the processing of larger datasets.



```

ASSISTANT: 📧 Subject: Request for Feedback on Quarterly Financial Report

Dear [Supervisor's Name],

I hope this email finds you well. I am writing to request your feedback on the Quarterly Financial Report that I prepared for our team. I would appreciate your thoughts on the report. Please let me know if there are any areas where I can improve or if there are any additional points you would like me to address. I am committed to delivering high-quality work. Thank you in advance for your time and input. I look forward to hearing from you soon.

Best regards,

[Your Name]

USER: Take a moment to evaluate and critique your own response.

ASSISTANT: 🧠 Self-Evaluation:

Overall, I believe the email I wrote is concise, clear, and professional. It effectively communicates the purpose of the email and requests specific feedback from my supervisor. However, I could have made the email even more effective by including more specific details about the report, such as its length, format, and any unique challenges I faced while preparing it. Additionally, I could have asked for feedback on other aspects of the report, such as its accuracy and completeness, to ensure that I have covered all relevant areas. Overall, I am satisfied with the email, but there is always room for improvement.

USER: Imagine you are writing a blog post comparing two popular smartphone models. Develop an outline for the blog post, including key points and subheadings to effectively con

```

**Figure 5. The result of running the code.**

It is noteworthy that due to the limited configuration of our PC—such as the speed of the hard disk, limited frequency and size of memory, and bottlenecks in GPU and CPU—we were unable to conduct some in-depth

experiments on performance and processing speed. However, by adjusting the preliminary parameters, `start_size`, `recent_size`, and `sample_size`, we were able to control the resource usage, processing speed, and the final relevance

of the results. We found that a larger `start_size` provides a more representative initial sample, which can be beneficial for establishing a baseline or for certain statistical analyses, albeit at the cost of slower start-up times. Increasing the `recent_size` can lead to more efficient processing of data in larger batches, potentially speeding up the training or inference process, but it may also require more memory and computational power. Utilizing a larger `sample_size` improves the reliability and robustness of the model's performance by reducing variance and providing a more accurate response. For specific tasks such as writing and reasoning, adjusting the `sample_size` significantly improved the performance (accuracy) of the answers compared to the original streaming algorithm.

### 7. Conclusion

The StreamingLLM is a very effective method to deal with the long text in a short conversation. However, it can't keep the accuracy when it faces the task that demand long-term memory and a great deal of data dependency. As a result, we add the reservoir sampling in the whole process to improve its accuracy. Then the computer will find more relative data to calculate the result. Through the experiment, we find the model's efficiency doesn't decline and the accuracy has improved.

## References

- [1] Praneeth Kacham, Vahab Mirrokni, Peilin Zhong, (2024, Mar17). PolySketchFormer: Fast Transformers via Sketching Polynomial Kernels, arXiv:2310.01655v3 [cs.LG]
- [2] Ali Jamali, Swalpa Kumar Roy, Avik Bhattacharya, Pedram Ghamisi, 2023, Local Window Attention Transformer for Polarimetric SAR Image Classification, IEEE Geoscience and Remote Sensing Letters, DOI: 10.1109/LGRS.2023.3239263
- [3] Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant Nair, Ilya Soloveychik, Purushotham Kamath, 2024, Keyformer: KV Cache reduction through key tokens selection for Efficient Generative Inference, Proceedings of Machine Learning and Systems 6(MLSys2024) Conference
- [4] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, Laurent Sifre, 2022, Training Compute-Optimal Large Language Models, arXiv: 2203.15556 [cs.CL]
- [5] Richard Startin, 2020, Reservoir Sampling, Reservoir Sampling | Richard Startin's Blog
- [6] Rajesh Jayaram, Gokarna Sharma, Srikanta Tirthapura, David P. Woodruff, 2019, Weighted Reservoir Sampling from Distributed Streams, Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Pages 218 - 235
- [7] Han, I., Jayaram, R., Karbasi, A., Mirrokni, V., Woodruff, D. P., & Zandieh, A. (2023, October 9). HyperAttention: Long-context attention in Near-Linear time. arXiv.org. <https://arxiv.org/abs/2310.05869>
- [8] Xiao, G., Tian, Y., Chen, B., Han, S., & Lewis, M. (2023, September 29). Efficient Streaming Language Models with Attention Sinks. arXiv.org. <https://arxiv.org/abs/2309.17453>
- [9] Mohammed Al-Kateb, Byung Suk Lee, X. Sean Wang, 2007, Adaptive-Size Reservoir Sampling over Data Streams, 19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)
- [10] Jagbir Kaur, 2023, STREAMING DATA ANALYTICS: CHALLENGES AND OPPORTUNITIES, International Journal of Applied Engineering & Technology, ISSN: 2633-4828