KV-X: Optimizing Memory Efficiency in LLMs for Long-Sequence Inference

Tarun Anoop Sharma
University Of Illinois Urbana-Champaign
Illinois, United States
{tsharma7} @illinois.edu

Abstract

Large Language Models (LLMs) have been widely used in real-time applications, such as chatbots. However, enabling LLM deployment on resource-constrained GPUs are challenging. This paper proposes a dual-strategy cache compression algorithm to improve memory efficiency for LLMs. Specifically, we target reducing the linear memory growth in KV cache allocation, without training specific attention heads that can attend to pre-determined important tokens. We selectively retain critical tokens in each cache line and dynamically reduce size of cache line across transformer layers to exploit information funneling in the cache. We develop our novel algorithm for selectively attending tokens in the less position sensitive (middle) part of the cache based on the current query and combine it with lambda-shaped static masking to retain sink and streaming tokens. The deliverables of this research include the development of a novel plug-and-play algorithm to reduce memory usage during large language model inference while maintaining generation accuracy so that constrained GPUs can faciliate longer sequence inferences.

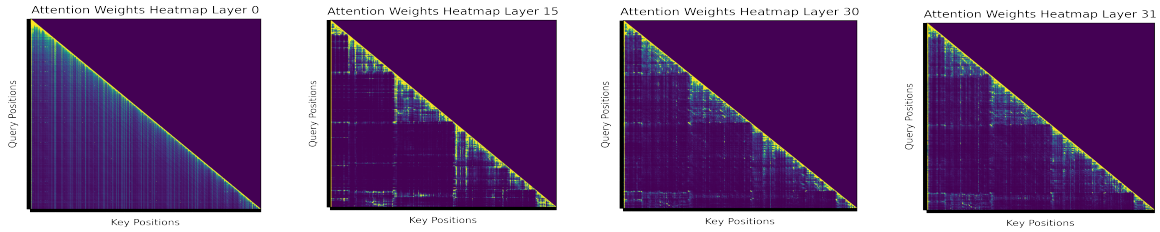
1 Introduction

The widespread adoption of Large Language Models (LLMs) in real-time applications, such as chatbots, virtual assistants, and content generation, underscores their transformative potential across various domains. However, deploying LLMs effectively, particularly on resource-constrained devices, presents significant challenges due to their substantial computational and memory requirements. One of the critical bottlenecks is the quadratic growth in memory consumption during inference, driven by the key-value (KV) cache allocations in transformer-based architectures. This growth not only limits latency but also increases resource demands, especially in long-sequence processing. Addressing these challenges without retraining and compromising model accuracy is paramount for broader accessibility and practical deployment of LLMs.

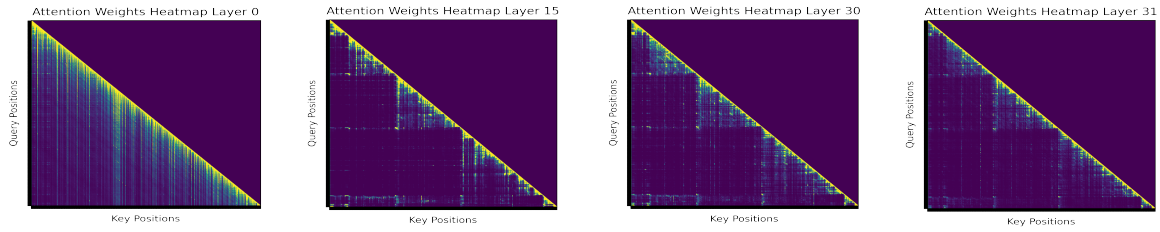
In autoregressive decoding, the KV cache plays a central role in enabling efficient attention mechanisms by storing intermediate results from earlier tokens for reuse in subsequent computations. However, as sequence lengths grow, the memory footprint of KV caches increases linearly, exacerbating resource demands. Existing solutions to alleviate this issue often require quantization (Jacob, 2018) techniques, which can negatively impact model performance or inherently maintain sparsity resulting in inefficient use of scarce GPU memory. This paper introduces a novel cache compression approach to dynamically improve compress KV cache footprint, targeting memory reduction specificaly for consumer GPUs that have small on-device memory capacity. Unlike similar works in literature, our solution does not require retraining attention heads with larger GPU resources, such as DuoAttention[(Xiao, 2024)].

Our work builds on the observation that attention patterns within decoder layers evolve slowly, suggesting important token propagation across cache layers that is described as information funneling. To save memory by exploiting this observation, we maintain gradually shrinking cache lines for layers. Our approach incorporates these top-K middle tokens based on their contribution to the current query, combined with static masking to further enhance memory efficiency. We also retain sink and streaming tokens that are known to effectively contribute to accurate and relevant text generations.

Through extensive evaluations across multiple model families, including Mistral, and Llama[(lla)], we demonstrate the generalizability and efficacy of our solution in reducing memory usage without sacrificing generation accuracy. The proposed algorithm represents a significant step toward enabling efficient LLM inference on resource-constrained



(a) Mistral-7B-Instruct-v0.3



(b) Llama-3.1-8B

Figure 1: We test attention patterns of retrieval-augmented generation across decoding layers with various models. It can be noticed that Mistral-7B-Instruct-v0.3 and Llama-3.1-8B does not have a clear information funneling pattern, namely "Masssive Attention" defined by PyramidKV(Cai, 2024). Final important tokens in the last layers that directly contribute to generation are mainly sink and streaming tokens for first two rows.

devices, paving the way for more accessible AI solutions.

2 Background and Motivation

The primary goal of this project is to develop an optimized memory management system for large language models (LLMs) to improve memory efficiency during long-sequence inference. As LLMs grow in memory to handle longer sequences and more complex tasks, memory management becomes a bottleneck, particularly on resource-constrained devices like GPUs. Addressing this challenge is critical for enabling broader deployment of LLMs in real-world applications, such as chatbots, virtual assistants, and content generation systems.

In autoregressive decoding, the key-value (KV) cache stores intermediate results from previously generated tokens, allowing the model to efficiently compute attention mechanisms without reprocessing the entire sequence. While this mechanism accelerates inference, it introduces quadratic memory growth with respect to sequence length. This exponential memory demand poses a significant challenge for deploying LLMs on GPUs with limited memory. For example, maintaining a KV cache for a 100K-token context in models like LLama2-7B requires over 50 GB of memory, far exceeding the capacity of most GPUs.

Existing solutions attempt to address these challenges through various strategies. LM-Infinite(Han et al., 2024) reports an important problem with KV Cache compression: some tokens that contribute to generation are lost when positioned in the middle of the processed sequences. To illustrate this problem, we evaluated LM-Infinite across different depths. For this experiment, we used Paul Graham's essays as the context and a fixed query needle: *"The best thing to do in San Francisco is eat a sandwich and sit in Dolores Park on a sunny day."* We prompted the LLM with the retrieval question: *"What is the best thing to do in San Francisco? Here is the most relevant sentence in the context:"*. We measured the average accuracy across a range of depth buckets to assess performance variations. Figure 2 shows the results for such varying depths, demonstrating the degrading accuracy when the needle is in the middle. Similarly, quantization techniques (Jacob et al., 2017) reduce the size of KV cache entries and model parameters but do not address the inherent sparsity of the cache, which impacts inference performance.

Compressing KV cache has been a widely adopted solution for addressing the sparsity in KV cache. For example, PyramidKV (Cai et al., 2024) introduces dynamic KV cache compression by leveraging the concept of Pyramidal Information Funneling, where attention becomes progressively focused across transformer layers. This approach effectively reduces memory usage in models such as LLama2 (Touvron et al., 2023) while maintaining accuracy. However, PyramidKV's effectiveness

is constrained to specific models and lacks generalization to other architectures like Mistral (Jiang et al., 2023) as can be observed in Fig 1. Our algorithm aims to capture attention funneling in such models by combining lambda-shaped mask introduced in LM-Infinite (Han et al., 2024) over sink and streaming tokens, along with adopting PyramidKV (Cai, 2024)'s dynamic cache line adjustments.

Compressed and static-sized cache techniques, such as those employed in SnapKV (Li et al., 2024) and LM-Infinite (Han et al., 2024), limit memory usage by retaining only a fixed set of tokens deemed critical, such as those near the beginning or end of the sequence. While these methods improve memory efficiency, they often fail to account for dynamically reducing cache line size that further optimizes the memory in situations where the tokens are funneled across decoder layers. Similarly, Heavy Hitter Oracle (Zhang, 2023) introduces considerable latency overhead into compression solution while actively evicting from the cache at runtime.

An ideal solution we propose is to enable LLMs to handle longer sequences on memory-limited GPUs without retraining or sacrificing performance. This would facilitate their deployment in real-time applications and on cost-effective hardware. By focusing on selective token retention and dynamic cache resizing, our work aims to bridge this gap, enabling broader adoption of LLMs across diverse domains and devices.

3 Algorithm

This section presents our dual-strategy cache compression algorithm, designed to optimize memory efficiency for large language models (LLMs) during inference that does not require any re-training. By dynamically allocating memory resources and selectively retaining critical tokens, the algorithm addresses the challenges of linear memory growth in key-value (KV) cache storage while preserving model accuracy.

3.1 Cache Initialization

Let L denote the number of transformer layers in the model, and C_{total} represent the total memory budget for the KV cache. The algorithm dynamically assigns layer-specific cache budgets, C_l, based on the varying importance of tokens across the transformer layers.

We initialize the total memory budget:

$$C_{\text{total}} = \sum_{l=1}^{L} C_l, \tag{1}$$

where C_l is the cache size for layer l. Drawing inspiration from Pyramidal Information Funneling introduced in PyramidKV[(Cai, 2024)], we allocate more memory at the lower layers (C_1) of cache and progressively decrease it in the higher layers. The cache budget for each layer follows an arithmetic progression:

$$C_l = C_L + (C_1 - C_L) \cdot \frac{L - l}{L - 1}, \tag{2}$$

where C_1 and C_L are the largest and smallest cache sizes allocated to the bottom and top layers, respectively.

3.2 Token Importance Assessment

At each layer l, the algorithm evaluates the importance of tokens by computing attention scores. Drawing inspiration from SnapKV (Li et al., 2024) observation on consistent attention allocation patterns across decoder layers, we specifically use the last window of the input sequence to compute attention over input prefix.

$$A^l = \text{softmax}\left(Q^l \cdot (K^l)^\top / \sqrt{d_k}\right), \tag{3}$$

where $Q^l, K^l \in R^{n \times d_k}$ are the query and key matrices for n tokens, and d_k is the key dimension. Tokens receiving higher cumulative attention from the current query are deemed critical for retention. Specifically, the importance score for the i-th token is calculated as:

$$s_i^l = \sum_{j=n-\alpha}^{n} A_{ij}^l, \tag{4}$$

where the summation aggregates the attention weights from the most recent α tokens (determined by cache line capacity, C_l), capturing the contribution of each token to the current query context, where ω represents the width of lambda-shaped mask as hyperparameter in our solution.

$$C_{l_{sink}} = C_{l_{streaming}} = \frac{\alpha}{2\omega} \tag{5}$$

represents the width of sink and streaming tokens, which sum up to total mask shape that is fractioned over cache line capacity by the hyperparameter.

$$C_{l_k} = C_l - (C_{l_{sink}} + C_{l_{streaming}}) \tag{6}$$

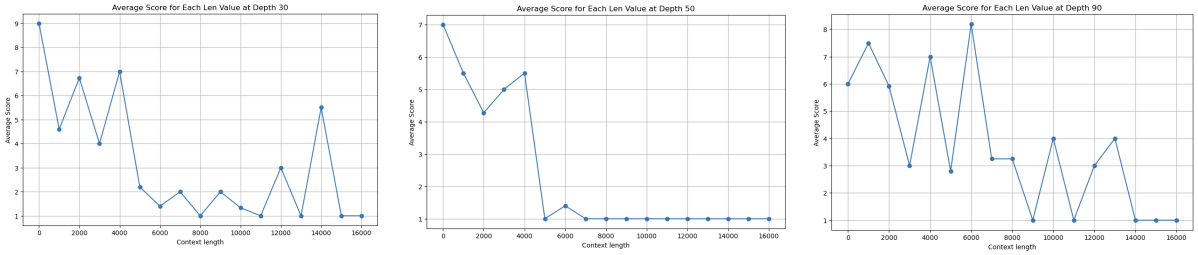


Figure 2: Demonstrating Lost in the Middle (Liu, 2024) problem in our case study for various needle depths with vanilla Llama-2-7B. Total accuracy is higher if needle is inserted under lambda-mask that can be adjusted to cover tokens in Depth %30 and %90. However, accuracy is not perfect under all context length settings when positioning the needle under such depths.

denotes the cache line capacity reserved for middle tokens that are sample with top-k.

Using the computed importance scores for middle tokens, the top C_{l_k} tokens with the highest scores are selected for retention in K^l, V^l:

$$K^l_{\text{selected}}, V^l_{\text{selected}} \in R^{C_l \times d_k}. \tag{7}$$

This dynamic selection preserves tokens most relevant to ongoing computations while discarding less informative entries, thereby reducing the memory footprint. For best memory efficiency, we compress the KV cache lines with this method both in prefilling and autoregressive decoding phases. We rely on top-k sampling for middle parts of the cache due to important token positions varying with input sequence length and therefore being unpredictable without training. Therefore, top-k sampling is efficient in reducing memory and also preserving accuracy.

3.3 Static Masking for Sink and Streaming Tokens

To further optimize memory allocation, we implement static masking to preserve specific token groups crucial for model accuracy:

- **Sink Tokens**: Tokens at the beginning of the sequence are retained to maintain global context.
- **Streaming Tokens**: Tokens near the end of the sequence are retained to ensure fidelity in the immediate task.

The static mask is defined as:

$$M^l = \{T_{\text{sink}}, T_{\text{streaming}}\}, \tag{8}$$

by using Equation 5. where T_{sink} and $T_{\text{streaming}}$ denote the indices of the sink and streaming tokens, respectively. Our algorithm uses static mask width as an important hyper-parameter to observe the tradeoff between accuracy and resource use optimization.

The reduced KV cache, $K^l_{\text{selected}}, V^l_{\text{selected}}$, is propagated across layers, dynamically adapting to the information density at each level. This strategy ensures compatibility with the downstream attention mechanisms and minimizes memory usage during long-sequence inference.

3.4 GPU to CPU Offloading

Our method utilizes CPU offloading for attention scores, masks and weights in the attention unit to free resource-constrained GPU memory with introduced latency overhead from our solution.

4 Results

We explore how to improve memory utilization on resource-constrained GPUs, therefore we pick Llama-3.1-8B-Instruct to fit on 24 GB GPU without quantization. We ignore quantization to refrain from impacted accuracy observations, due to already expected accuracy degradation from KV cache compression. Furthermore, Llama-3.1-8B-Instruct also exhibits similar attention patterns to shown heatmaps in Figure 1 that did not have exploitable information funneling patterns. Therefore, it manifests as a good fit for evaluating efficiency of our new approach.

Our compression method is enabled when the sequence length in KV cache exceeds 256, 512, or 1024 tokens. In other words, our solution is performed when the input sequence length exceeds the enforced context limit length. Therefore, we vary context limit while fixing the input sequence length of 1024, 2048 and 4096 that can fit on 24 GPUs. We implement this by imposing a custom

Lambda Mask Total Width	Sequence Length	Enforced Context Limit	Result	Compressed Memory
$\frac{1}{2}$*Cache Line	1024	512	F	18.0 GB
	2048	1024	P	21.7 GB
	2048	512	P	19.8 GB
	2048	256	P	18.9 GB
	4096	1024	-	OOM
	4096	512	F	23.4 GB
	10000	-	-	OOM
$\frac{1}{4}$*Cache Line	1024	512	P	17.4 GB
	2048	1024	P	19.9 GB
	2048	512	P	18.9 GB
	2048	256	P	18.5 GB
	4096	1024	F	23.6 GB
	4096	512	F	21.8 GB
$\frac{1}{8}$*Cache Line	1024	512	F	17.3 GB
	2048	1024	F	18.9 GB
	2048	512	P	18.5 GB
	2048	256	F	18.3 GB
	4096	1024	F	21.6 GB
	4096	512	P	20.9 GB
$\frac{1}{16}$*Cache Line	1024	512	F	17.2 GB
	2048	1024	F	18.5 GB
	2048	512	P	18.3 GB
	2048	256	P	18.2 GB
	4096	1024	P	21 GB
	4096	512	P	20.6 GB
$\frac{1}{32}$*Cache Line	1024	512	F	17.1 GB
	2048	1024	P	18.3 GB
	2048	512	P	18.3 GB
	2048	256	F	18.2 GB
	4096	1024	F	20.6 GB
	4096	512	F	20.4 GB

Table 1: Algorithm results on Needle-in-Haystack benchmark with "Pass (P)" and "Fail (F)" cases. Tested on Nvidia RTX4090 GPU 24 GB with depth percent %100 and maximum generation length limit of 256 tokens enforced on Llama-3.1-8B-Instruct

context limit to the model to control upper limit on such KV cache allocations. We assume given user input is always longer than the model output to demonstrate the requirement for compressing the KV cache on constrained GPUs.

Table 1 shows algorithm evaluation on Needle-in-Haystack benchmark, where we use a strict evaluation on accuracy, which expects the whole needle in the output to pass the test.

We acknowledge the Lost-in-the-Middle [(Liu, 2024)] problem and expect the model to attend to the needle sequence by placing it on the less position-sensitive part within the lambda-shaped static mask. Particularly, we use depth percent %100 in all of our testcases for this purpose. Although we attempt to make sure needle is attended, in some compression scenarios model fails to pass the test. In other words, we still observe lost accuracy in some cases due to losing information.

Cache line size is an adaptive parameter set by our algorithm as explained in 3.1. For example, when static mask width is one fourth of the total cache line, with cache line of 2048 tokens, 256 tokens from the left (sink tokens) and 256 tokens from the right of the line (streaming tokens) will be attended. Remaining 1536 tokens will be reduced to 256 tokens by selecting top 256 of them from the middle of the line. In many cases, mask

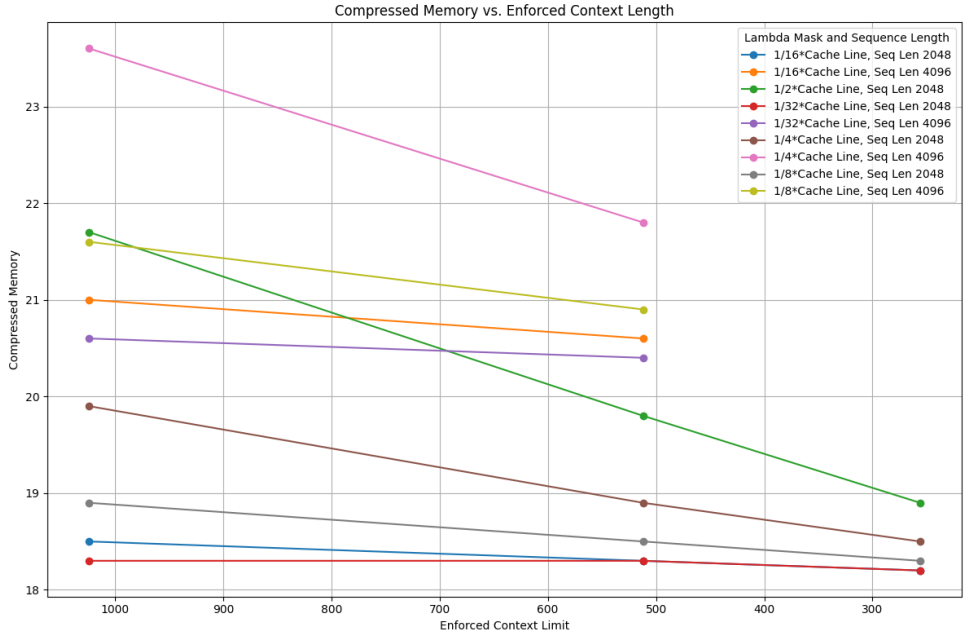


Figure 3: Total memory under various Lambda-Mask widths (in terms of Cache line capacity) and Input Sequence Lengths

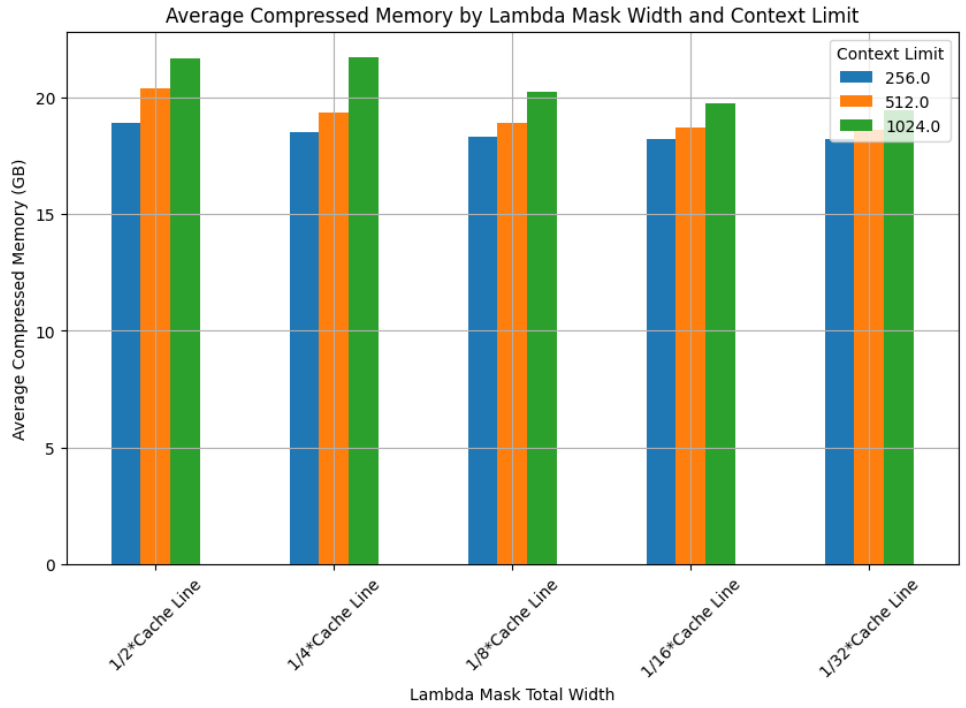


Figure 4: Total memory for each Lambda-Mask Width, which suggests better compression rates as lambda-mask that includes sink and streaming tokens into attention becomes narrow

width does not need to be as wide as 256 tokens to capture the important tokens that contribute to the accurate answer. Our experiments aim to show that narrow mask widths are functional in saving memory allocations and sufficient in offering accurate inference.

The key observations from our results are summarized as follows.

- Since KV cache needs to be compressed after the sequence length exceeds the controlled context limit, as the context limit decreases in Table 1, our solution performs higher rates

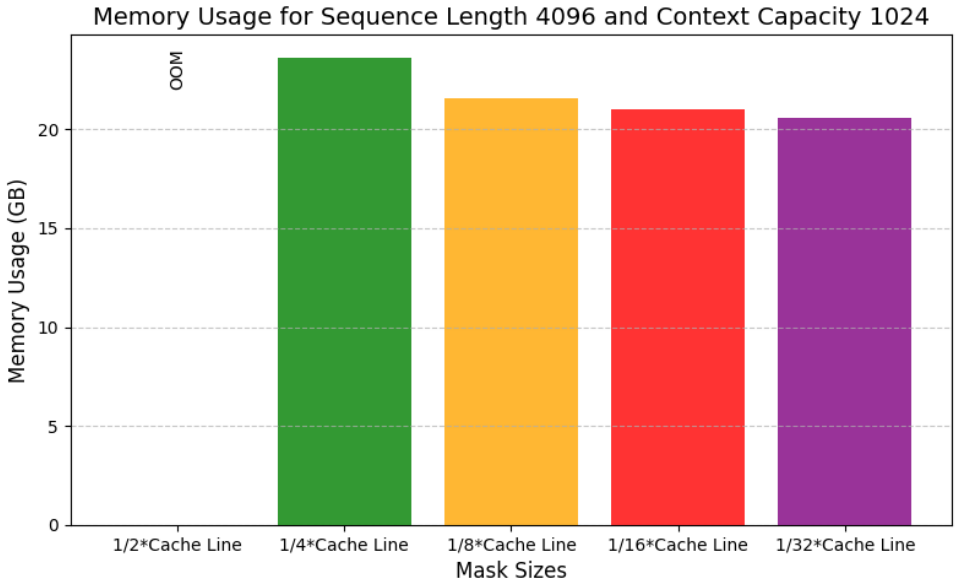


Figure 5: Total memory with Input Sequence Length 4096 and Context Capacity of 1024 tokens, demonstrating the resolution of Out Of Memory errors with our method

of compression with an observable decrease in total memory allocations on GPU. Figure 3 shows the change in Compressed Memory as Enforced Context Limit Decreases for different Lambda Mask and Sequence Length pairs.

- Adjusting the total width of the mask is crucial in our solution since it has direct impacts on the generation accuracy. As the mask becomes more narrow, captured information across cache layers are well maintained with optimized memory, compared to larger mask sizes that typically require larger memory space.

- Across all experiments, best accuracy and optimization combination seems to be Sequence length 2048 and Context Limit 512 that facilitates accurate inference on the consumer GPU running relatively smaller models. This combination succeeds under various mask adjustments.

- We are able to unlock LLM generation without erros when using smaller mask sizes compared to initial OOM scenarios, such as context length=4096 and context capacity=1024 and 512.

5 Team Contribution

Selin, the project leader, guided the team by defining the problem and solutions and determining the datasets and approach. She developed, refined, and tested the algorithm, debugged issues in the needle-in-haystack benchmark, and provided the PyramidKV baseline in Colab. She also led the implementation of the project's milestones, which included benchmarking Needle in Haystack (Tarun), benchmarking PyramidKV (Manvi), and implementing the algorithm based on insights from these milestones.

Tarun explored various benchmarks suitable for the project and prepared the needle-in-haystack benchmark. He also benchmarked static masks on the Needle in a Haystack on multiple LLMs and investigated the low accuracy at Depth 0 for LM-Infinite. He reviewed the literature to brainstorm together the project's idea, the pivot and participated in team discussions throughout the project timeline. He contributed to the preparation of presentations, and reports throughout the project's duration.

Manvi actively participated in discussions on algorithm development, offering ideas and researching existing methods to identify potential improvements for the proposed algorithm. she worked on benchmarking PyramidKV across various LLMs and ran experiments to evaluate PyramidKV's performance and suggest optimizations for the proposed method. She helped create presentations, and contributed to documentation. She worked closely with the team to troubleshoot challenges and refine benchmarks.

6 Limitations and Future Work

The proposed algorithm, introduces latency overhead during compression computations in CPU and communication overhead between GPU and CPU. Furthermore, current int64 computations introduce %5 memory overhead over the original model inference. Our next direction is to ensure existing CPU offloading approach helps actually prevent this memory overhead. Furthermore, although we have positioned the needle on lambda-shaped mask in attention, we still observed failing testcases that dropped the accuracy, in alignment with the motivational results that show LLMs inherent bottlenecks. These limitations highlight requirement of further refinement and testing to enhance our algorithm's generalizability and robustness. We will continue working on our optimization methodology.

References

https://huggingface.co/meta-llama/meta-llama-3-8b.

Zefan Cai et al. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling.

Zhang Y. Gao B. Liu Y. Liu T. Lu K. ... Xiao W. Cai, Z. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling.

Chi Han et al. 2024. LM-infinite: Zero-shot extreme length generalization for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008, Mexico City, Mexico. Association for Computational Linguistics.

Benoit Jacob et al. 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference.

Kligys S. Chen B. Zhu M. Tang M. Howard A. ... Kalenichenko D. Jacob, B. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *In Proceedings of the IEEE conference on computer vision and pattern recognition*.

Albert Q. Jiang et al. 2023. Mistral 7b.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation.

Lin K. Hewitt J. Paranjape A. Bevilacqua M. Petroni F. Liang P. Liu, N. F. 2024. Lost in the middle: How language models use long contexts. In *Transactions of the Association for Computational Linguistics*.

Hugo Touvron et al. 2023. Llama 2: Open foundation and fine-tuned chat models.

Tang J. Zuo J. Guo J. Yang S. Tang H. ... Han S. Xiao, G. 2024. Duoattention: Efficient long-context llm inference with retrieval and streaming heads.

Sheng Y. Zhou T. Chen T. Zheng L. Cai R. ... Chen B. Zhang, Z. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*.