Memory Reduction for LLM Inference via KV-Cache Compression

Jai Jain, Shantanu Patankar, Neelabh Sinha, Chaitra Sathe
Georgia Institute of Technology
{jjain47, spatankar34, nsinha68, csathe6}@gatech.edu

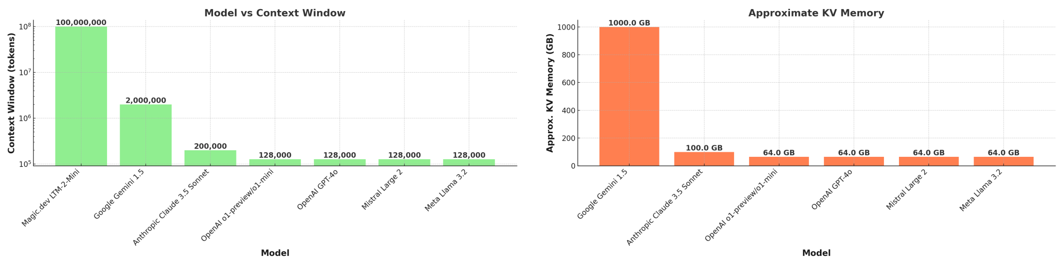
Abstract

Large Language Models (LLMs) have demonstrated exceptional capabilities across various applications, particularly in long-context tasks. However, extending context lengths significantly increases memory demands, with the Key-Value (KV) cache emerging as a critical bottleneck. Current GPUs often lack the capacity to support the memory requirements of state-of-the-art LLMs for long-context processing. Existing KV cache compression techniques, such as eviction-based methods and quantization, often compromise global context retention or computational efficiency. To address these limitations, we propose a novel KV cache compression approach that emphasizes merging over eviction. Our method introduces a coarse-grained compression strategy that progressively merges older tokens using pooling techniques while preserving recent tokens with higher fidelity. Additionally, we leverage hierarchical attention patterns in transformers to implement a pyramidal layer compression strategy, which reduces memory usage by progressively decreasing the compression threshold across layers. This dual approach retains critical global context in lower layers while focusing on localized information in higher layers. Extensive experiments on long-context tasks from the Longbench benchmark demonstrate that our method achieves competitive performance compared to state-of-the-art approaches while significantly reducing memory consumption. These results highlight the potential of our approach to enable efficient LLM inference in memory-constrained environments without sacrificing generation quality.

1 Introduction

Large Language Models (LLMs) have exhibited remarkable capabilities across a wide array of applications, particularly in long-context tasks that are increasingly critical in real-world scenarios. Recent advances in state-of-the-art LLMs have focused on scaling models to handle extended contexts effectively. Notable examples include OpenAI's ChatGPT Achiam et al. [2023], Anthropic's Claude Anthropic [2024], Meta's LLaMA-3 Van Der Maaten et al. [2024], Mistral Team [2024b], and Google's Gemini Pro-1.5 Team [2024a], which supports a staggering context length of 1M tokens. However, enabling such long-context processing poses significant challenges in terms of memory and computational efficiency, with the *Key-Value (KV) Cache* emerging as a primary bottleneck.

The KV cache is integral to LLMs, storing the key and value states of the past sequence for calculation of attention scores of the current token during autoregressive decoding. Reusing these states significantly enhances computational efficiency at the cost of memory. As the sequence length increases, the memory required for the KV cache scales linearly with the context length and batch size. For instance, a 175 billion parameter GPT-3 model requires approximately 1,208 GB of GPU memory for a batch size of 64 and a sequence length of 4,096 tokens, far exceeding the capacity of even the most advanced GPUs. Figure 1 shows the context size limit and corresponding memory requirement of some of the SOTA LLMs. With most of the GPUs having 80 GB memory availability, this size can't



(a) Context Window

(b) Cache Memory Requirement

Figure 1: Context window and corresponding KV Cache budget required for some of the current state-of-the-art LLMs

be supported. Memory-constrained environments and mobile device inference, therefore, can't utilize this huge growing potential of LLMs for long-context tasks as is. This increasing memory demand underscores the need for effective KV cache compression techniques that preserve generation quality while supporting long contexts.

Existing approaches to KV cache compression primarily target one of three dimensions: sequence length, number of layers, or number of attention heads, as also shown in Figure 2. Techniques such as merging or eviction have been explored, with quantization offering an orthogonal strategy that can complement these methods. For example, *StreamingLLM* Xiao et al. [2023] maintains a constant KV cache budget using fixed "sink" tokens and a sliding context window, allowing for infinite sequence generation. However, this approach struggles with long-context tasks, where retaining global context is critical. Similarly, eviction-based methods such as *H2O* Zhang et al. [2023] discard older tokens to manage memory. Although effective for short-context tasks, these methods risk irreversibly removing vital information, leading to degraded performance on long-context tasks. This limitation is akin to erasing the answer before asking a question.

Recent insights from works such as *PyramidKV* Cai et al. [2024] reveal that lower layers tend to attend to global context, while higher layers focus on more localized information, forming a pyramid-like attention pattern across layers. Inspired by these findings, methods such as *Duo Attention* Xiao et al. [2024] and *Razor Attention* Tang et al. [2024] have introduced learnable mechanisms to identify and prioritize retrieval heads. However, existing strategies still leave room for improvement in maintaining long-context fidelity while reducing memory consumption.

In this work, we propose a novel approach to KV cache compression that prioritizes *merging* over *eviction*. Similar to existing works, we retain attention sinks and local window of key-values, which are deemed important Xiao et al. [2023]. However, unlike eviction-based methods, we also retain important global context by merging the tokens using pooling strategies. Specifically, we introduce a coarse-grained compression strategy of global context that progressively merges tokens once a user-defined threshold is reached. The key-values of the oldest tokens are compressed more aggressively, while the most recent tokens are preserved with higher fidelity, so that the key information is still intact. Additionally, motivated by findings that the initial layers of transformers attend to tokens more globally across the context, gradually localizing in the deeper layers Cai et al. [2024], we introduce a method for linear decrease of the compression threshold across layers of transformers to further save memory while retaining important context.

Our contributions can be summarized as follows:

- We propose a novel method for reducing the memory footprint of the KV cache via a two-dimensional merging strategy across sequence length and layers of the LLMs.
- We develop a methodology that iteratively merges the least recently generated tokens after a user-defined threshold is reached, preserving relevant global context while reducing memory.
- We introduce linear compression of the threshold along the layers to allow additional memory efficiency, alongside enabling more local focus in the deeper layer of transformers.
- We evaluate our method on long-context tasks and demonstrate its effectiveness compared to state-of-the-art approaches.

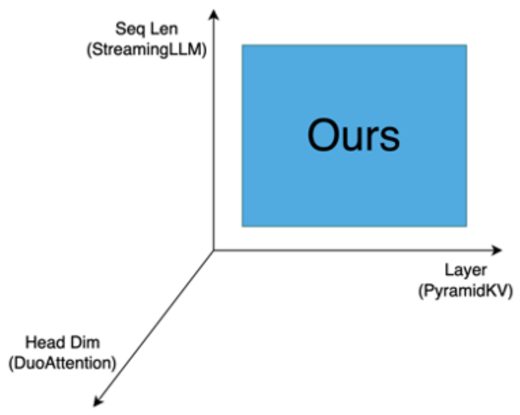


Figure 2: Three possible axes on which KV Cache can be compressed, followed by the dimensions in which our method achieves compression.

2 Related Work

Key-value cache compression is a critical area of research aimed at optimizing the memory and computational efficiency of large language models during inference. Existing methods address this challenge across three axes: sequence length, attention heads, and layers, employing techniques broadly categorized into *eviction* and *merging*. Below, we summarize the most relevant contributions in each category.

2.1 Sequence Length

Eviction Methods: Several works focus on reducing the sequence length of the KV cache by evicting less significant tokens. H2O Zhang et al. [2023]identifies "heavy-hitter" tokens that contribute most to attention scores and evicts the remaining tokens greedily . StreamingLLM Xiao et al. [2023] employs a sliding context window and emphasizes the importance of "sink tokens" to maintain relevant information.

Merging Methods: Unlike eviction, merging methods attempt to reduce the KV cache size by combining similar key-value states. KVMerger Wang et al. [2024] utilizes a Gaussian kernel-based weighted merging mechanism to identify and merge similar tokens dynamically . CaM Zhang et al.pioneers token-level merging, opting to merge instead of evicting tokens in the cache. CacheBlend Yao et al. [2024] innovatively fuses pre-computed KV caches by selectively recomputing a small subset of tokens, irrespective of whether the prefixes match.

2.2 Attention Heads

Eviction Methods: Techniques targeting attention heads aim to retain only the most crucial information. Duo Attention Xiao et al. [2024] and Razor Attention Tang et al. [2024] categorize attention heads as retrieval and streaming heads, enabling efficient compression by focusing on local or global contexts. Ada-KV Feng et al. [2024] adaptively allocates a compression budget to each head based on its attention distribution , while FastGen Ge et al. [2023] dynamically selects optimal strategies for different attention patterns . SnapKV Li et al. [2024] employs a voting mechanism within a sliding observation window to identify and retain important tokens per head.

Merging Methods: Admixture of Heads Nguyen et al. [2022] reduces computation by representing H individual heads as mixtures of M global heads ($M < H$), enabling shared representation and reducing redundancy across heads.

2.3 Layers

Eviction Methods: Layer-based eviction strategies aim to dynamically compress KV caches across model layers. D2O Wan et al. [2024] adjusts eviction thresholds per layer and combines token eviction with dynamic token merging. PyramidKV Cai et al. [2024] progressively reduces the context size in higher layers, allowing lower layers to attend to all tokens while restricting higher layers to local contexts. PyramidInfer Yang et al. [2024] introduces a layer-by-layer compression approach that retains only the most influential keys and values (termed "pivotal context") by leveraging recent attention consistency across layers.

Merging Methods: Layer-level merging seeks to exploit the redundancy in KV states between adjacent layers. MLKV Zuhri et al. [2024] introduces a shared KV cache across layers to minimize memory usage. MiniCache Liu et al. [2024] compresses the KV cache by identifying and merging highly similar states in middle-to-deep layers using a reparameterization approach that interpolates directions while preserving magnitude, ensuring critical distinctions are retained.

3 Methodology

Our methodology for KV cache compression combines two complementary techniques: *Pyramidal Layer Compression* and *Sequence Length Compression*. These approaches are motivated by the hierarchical attention patterns observed in large language models (LLMs) Cai et al. [2024] and aim to optimize memory usage while preserving essential contextual information. Below, we describe each technique in detail.

3.1 Pyramidal Layer Compression

The *Pyramidal Layer Compression* method leverages the observation that lower layers in transformers attend to global context, while higher layers focus on localized information Liu et al. [2024] Cai et al. [2024]. To exploit this, we progressively reduce the KV cache size across layers, forming a pyramid-like structure.

Compression Strategy Let L_0 be the initial cache budget for the 0th layer of an m-layer transformer decoder, and β be the compression ratio between the 0th and $(m-1)$-th layer. The cache budget for the $(m-1)$-th layer is defined as:

$$L_{m-1} = \frac{L_0}{\beta}.$$

Assuming a linear rate of decrease in window size across layers, the cache size for the k-th layer is given by:

$$L_k = L_0 \left[1 + \frac{k}{m}\left(\frac{1}{\beta} - 1\right)\right]$$

With this approach, we reduce the total KV budget of layers from $m(L_0)$ to L^* such that

$$L^* = L_0(m-1)\left[\frac{1}{2} + \frac{1}{\beta}\right]$$

Key Considerations As shown in Figure 3, sink tokens, that is, initial tokens are not compressed across any dimension, ensuring their critical context is preserved. This approach introduces two hyper-parameters:

- Initial cache budget (L_0)
- Pyramid compression ratio (β)

These parameters allow for flexible control over memory savings and computational efficiency.

3.2 Sequence Length Compression

To further optimize memory usage, we apply sequence length compression by pooling key-value pairs within fixed window sizes. Previous work Xiao et al. [2023] Zhang et al. [2023] shows that the most recent tokens are highly critical to the context. Hence, in our compression method, we use a dynamic compression method where the least recent tokens are compressed as more decode tokens are generated, as shown in Figure 4. We use a fixed window size to merge the least recent tokens. Therefore, the window size is also a hyper-parameter that allows us to flexibly control how much context to compress, depending on the KV budget. We also experiment with different pooling (merging) mechanisms as explained below.

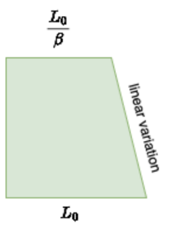


Figure 3: Illustration of Pyramidal Layer Compression. The cache budget decreases linearly across layers, preserving more global context in lower layers.

Pooling Mechanism As illustrated in Figure 4, key-value pairs are grouped into windows of a predefined size. Each window is then compressed into a single representation using one of the following pooling strategies:

1. Mean pooling: Computes the average of key-value pairs within a window.
2. Max pooling: Retains the most significant key-value pair in each window.
3. Best pooling: Dynamically selects key-value pairs based on their relevance to downstream tasks.

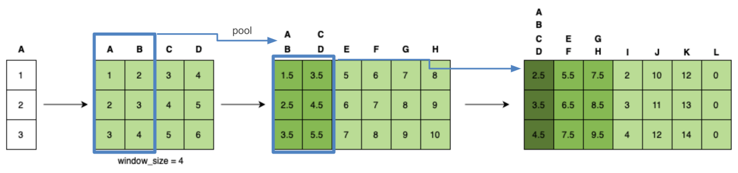


Figure 4: Illustration of Sequence Length Compression. Key-value pairs are pooled based on a fixed window size using different pooling strategies (e.g., mean, max, or best).

3.3 Motivation and Design Principles

Our design is motivated these observations (Figure 5):

1. Attention sinks (e.g., start tokens) consistently receive high attention scores but do not require compression.
2. Attention patterns become increasingly localized in higher layers, which justifies progressive compression along the layers.
3. There are few attention sinks apart from the initial and most recent tokens, which justifies compression along the sequence length.

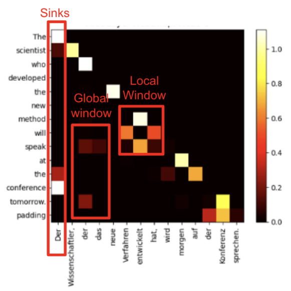
By combining pyramidal layer compression with sequence length compression, our methodology aims to significantly reduce the memory budget for KV cache while maintaining high generation quality.

4 Experiments

4.1 Experimental Setup

To evaluate the effectiveness of our proposed KV cache compression technique, we conducted experiments on five datasets from the Longbench benchmark Yuan et al. [2024]. These datasets cover diverse long-context tasks, ensuring a comprehensive assessment of performance. The tasks, evaluation metrics, and datasets are summarized in Table 1.

The datasets encompass both extractive and generative tasks with an average context length of 27k tokens per instance, enabling us to test the model's ability to handle extensive input contexts. We



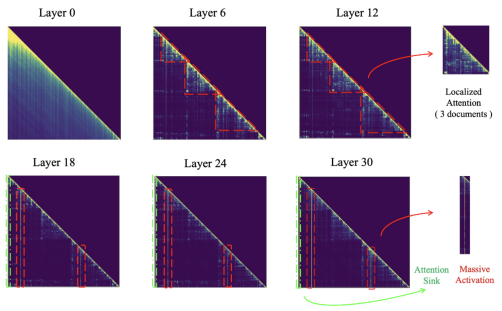


Figure 5: Motivation for KV Cache Compression. Left: Attention sinks and global/local windows in attention maps. Right: Localized attention patterns emerge as we move up transformer layers.

Dataset	**Task Type**	**Evaluation Metric**
Qasper	Single-document QA	F1 Score (0-100)
Musique	Multi-document QA	F1 Score (0-100)
Multinews	Summarization	ROUGE-L (0-100)
TREC	Few-shot Classification	Accuracy (0-100)
TriviaQA	Few-shot ODQA	F1 Score (0-100)

Table 1: Datasets and tasks used for evaluation.

employed Llama-3-8B-Instruct Dubey et al. [2024], a state-of-the-art large language model optimized for instruction-following tasks, as our base model for these experiments. Each task was evaluated using its respective metric to avoid bias from differing evaluation standards. Each task was evaluated using its respective metric to ensure a fair and accurate assessment aligned with the nature of the task. For instance, QA tasks utilized metrics like F1 scores whereas summarization tasks employed ROUGE-L scores. This approach avoids bias from differing evaluation standards while providing a comprehensive and multidimensional assessment of the model's ability to process long contexts effectively.

4.2 Implementation Details

Our method applies a threshold-based KV cache compression strategy with layer-specific thresholds that adapt to the pyramid-like attention pattern of the model. We implemented this strategy using PyTorch and performed all experiments on NVIDIA H100 GPUs with 80 GB of memory. The batch size for all tasks was set to 1, ensuring consistency across evaluations.

We conducted a series of experiments to evaluate our approach across various datasets and configurations. The experiments focused on analyzing the impact of pooling strategies, pyramid compression ratios, cache budgets, and comparing our method with state-of-the-art baselines.

We compared our proposed method against three baselines:

- **StreamingLLM** Xiao et al. [2023]: A sliding window-based approach with fixed "sink" tokens for context management. This method retains a small number of initial tokens (attention sinks) in the KV cache while discarding older tokens to make room for new ones, enabling efficient handling of sequences up to millions of tokens without requiring model retraining.
- **H2O** Zhang et al. [2023]: An eviction-based method that dynamically discards older tokens based on their importance, determined by accumulative attention scores. It balances the retention of recent and high-value tokens (heavy hitters) to optimize memory usage while maintaining model performance.
- **FullKV**: A strategy that retains all key-value (KV) cache entries throughout the sequence, ensuring no information is discarded. While this approach avoids any performance degra-

dation due to token loss, it is computationally and memory-intensive, especially for long sequences.

5 Results

5.1 Performance as a Function of Pooling Types

We evaluated the performance of three pooling strategies—Best, Mean, and Max—across five datasets: Multi-News, Musique, Qasper, TREC, and TriviaQA. The results, as shown in Figure 6, reveal distinct patterns based on the size of the key-value (KV) budget allocated for processing. For larger KV budgets, both Mean and Best pooling outperform Max pooling, suggesting that these methods are better at preserving contextual information when more resources are available. Conversely, for smaller KV budgets, Best pooling consistently outperforms Mean pooling, which in turn surpasses Max pooling in terms of performance.

For smaller KV budgets, we observed that tokens undergo multiple rounds of compression due to the limited memory available. This repeated compression can lead to significant information loss, as the merged tokens fail to retain sufficient contextual details from the original input. In such scenarios, Best pooling excels because it prioritizes the most important token within a group during each merge step, thereby mitigating the loss of critical information. On the other hand, Mean pooling averages contributions from all tokens in a group, which introduces some noise but still retains more context than Max pooling, which only considers the single most prominent feature and discards all others.

In contrast, for larger KV budgets where fewer merges are required per token for the same generation length, Mean pooling demonstrates superior performance compared to Max pooling. This is because averaging contributions from all tokens in a group allows Mean pooling to better preserve contextual nuances and distribute importance across multiple features. Max pooling's focus on extreme values becomes less effective in this scenario as it neglects subtler but potentially relevant details. These observations highlight how the choice of pooling strategy should be tailored to the available KV budget to optimize performance across different tasks and datasets.

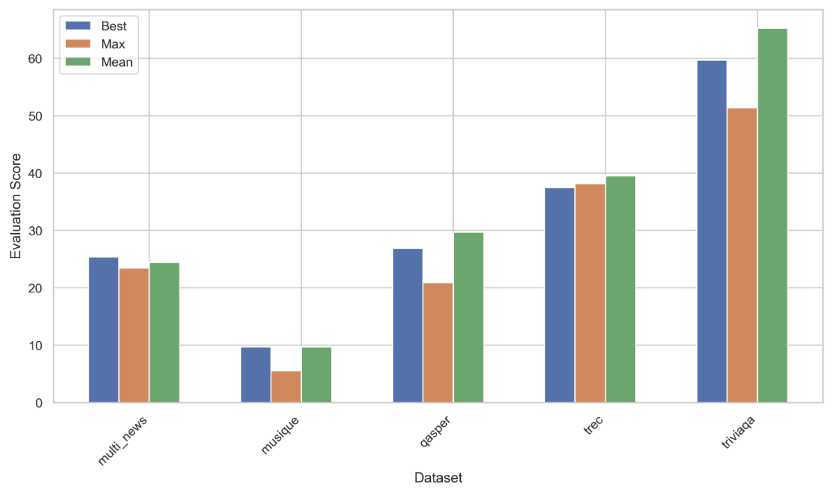


Figure 6: Performance as a function of pooling types across datasets. Initial cache budget = 4096, Pyramid compression ratio = 4, Sink tokens = 4.

5.2 Comparison with State-of-the-Art Methods

We compared our method against three baselines: **Full KV**, **Streaming LLM**, and **H2O** with results summarized in Table 2. We observed that our method achieves competitive performance relative to Streaming LLM and H2O, particularly on datasets such as Qasper and TREC.

However, we also observed a noticeable performance gap when compared to Full KV, which serves as an upper bound in terms of achievable performance. We hypothesize that this gap arises because Full KV retains all key-value pairs in memory, allowing it to fully leverage the input context without any compression or truncation. In contrast, our method operates under constrained memory budgets, which may limit its ability to capture long-range dependencies or nuanced details in the input.

To bridge this gap and further enhance our method's performance, we propose several potential improvements. These include increasing the context window size to process more information at once, recomputing positional embeddings to better align with compressed representations, and optimizing decoding strategies for more accurate output generation. These adjustments could help mitigate the trade-offs introduced by memory constraints and bring our method closer to the upper-bound performance demonstrated by Full KV.

Name	Qasper	Musique	Multi-News	TREC	TriviaQA
Full KV	29.75	22.35	26.21	73.00	90.56
Streaming LLM	8.68	15.97	14.64	38.00	84.70
H2O	11.34	17.94	19.13	38.00	72.30
Ours	14.05	6.95	6.00	33.33	23.61

Table 2: Comparison with state-of-the-art methods across datasets. Pooling = Best, Avg cache budget = 64, Sink tokens = 4.

5.3 Performance as a Function of Pyramid Compression Ratios

We analyzed how varying pyramid compression ratios (1, 2, 4, 8) affect performance, as shown in Figure 7. We observed that while performance degradation generally occurs with increasing compression ratios, the extent of this degradation remains within an acceptable range for most tasks. This suggests that the model retains a reasonable level of effectiveness even under higher compression settings.

Interestingly, we observed that summarization tasks, such as Multi-News, exhibit less performance degradation compared to question-answering (QA) tasks like TriviaQA. We hypothesize that this disparity arises because summarization tasks rely more on capturing the overall context and gist of the input, which can be preserved even with reduced information granularity. In contrast, QA tasks often require precise retrieval of specific details from the input text. As the pyramid compression ratio increases, the budget for local windows decreases, potentially leading to a loss of fine-grained information critical for answering detailed questions. This difference highlights how task-specific requirements influence the sensitivity to compression and underscores the importance of tailoring compression strategies to the nature of the task at hand.

5.4 Performance as a Function of Initial Cache Budget

We studied the impact of varying initial cache budgets (256, 512, 1024, 4096) on performance, as illustrated in (Figure 8). We observed that increasing the cache budget consistently improves performance across all datasets.

However, we observed that the magnitude of performance improvement varies across tasks and evaluation metrics. For example, TriviaQA benefits significantly from larger KV cache budgets compared to Musique. We hypothesize that this is because TriviaQA relies heavily on retrieving specific details from extended contexts, making it more sensitive to the availability of cached information. A larger KV cache allows more of these critical details to be stored and reused during inference, reducing computational load and improving accuracy. In contrast, Musique may involve less context-sensitive tasks or shorter input sequences, making it less dependent on extensive caching and thus showing diminishing returns with increased cache size.

These findings underscore the importance of tailoring KV cache budget allocation to task-specific requirements. Allocating larger budgets to tasks like TriviaQA that demand precise and contextually rich retrieval ensures optimal performance gains. Conversely, for tasks like Musique with lower sensitivity to cache size, smaller budgets may suffice, enabling more efficient resource utilization.

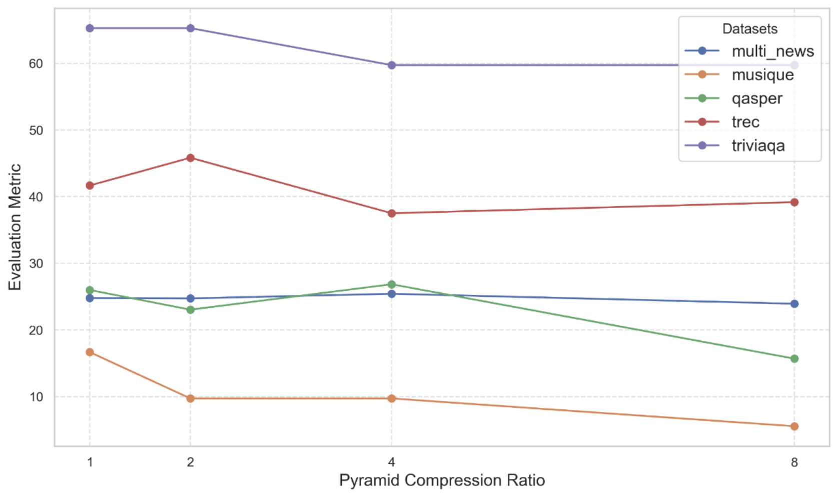


Figure 7: Performance as a function of pyramid compression ratio across datasets. Pooling = Best, Initial cache budget = 4096, Sink tokens = 4.

This highlights the nuanced trade-offs in KV cache management and emphasizes the need for dynamic or adaptive strategies to balance memory efficiency with task performance requirements.

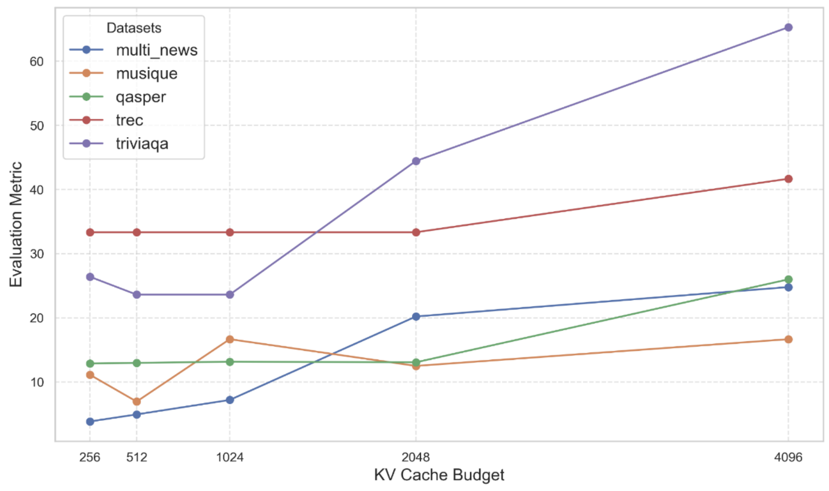


Figure 8: Performance as a function of initial cache budget across datasets. Pooling = Best, Pyramid compression ratio = 1 (rectangular), Sink tokens = 4.

6 Conclusion

In conclusion, this paper addresses the critical challenge of memory efficiency in large language models (LLMs) during long-context inference by proposing a novel Key-Value (KV) cache compression strategy. Our method emphasizes merging over eviction to preserve essential contextual information while reducing memory usage. Specifically, we introduce a threshold-based approach that dynamically adjusts compression levels across layers, aligning with the pyramid-like attention distribution observed in LLMs. This ensures that global context is retained in lower layers while higher layers focus on localized information.

Through extensive experiments on diverse long-context tasks from the Longbench benchmark, our approach demonstrates competitive performance against state-of-the-art methods like StreamingLLM and H2O. The results highlight the importance of selecting appropriate pooling strategies and optimizing compression parameters to balance memory efficiency and generation quality. While our method shows promise, there remains a performance gap compared to the Full KV baseline, which represents the upper bound of achievable results.

Future work could explore enhancements that refine positional embeddings to bridge this gap further. By addressing these areas, our approach can pave the way for more efficient and scalable LLMs capable of handling increasingly demanding applications requiring extended context lengths.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Anthropic. Claude, 2024. URL `https://www.anthropic.com`. Large language model.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *arXiv preprint arXiv:2407.11550*, 2024.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *arXiv preprint arXiv:2405.14366*, 2024.

Tan Nguyen, Tam Nguyen, Hai Do, Khai Nguyen, Vishwanath Saragadam, Minh Pham, Khuong Duy Nguyen, Nhat Ho, and Stanley Osher. Improving transformer with an admixture of attention heads. *Advances in neural information processing systems*, 35:27937–27952, 2022.

Hanlin Tang, Yang Lin, Jing Lin, Qingsen Han, Shikuan Hong, Yiwu Yao, and Gongyi Wang. Razorattention: Efficient kv cache compression through retrieval heads. *arXiv preprint arXiv:2407.15891*, 2024.

Gemini Team. Gemini: A comprehensive overview of the next-gen language model. *arXiv preprint arXiv:2409.00084*, 2024a.

Mistral Team. Mistral: A new era of language models. *arXiv preprint arXiv:2408.00001*, 2024b.

Laurens Van Der Maaten et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL `https://arxiv.org/abs/2407.21783`.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, and Mi Zhang. D2o: Dynamic discriminative operations for efficient generative inference of large language models. *arXiv preprint arXiv:2406.13035*, 2024.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*, 2024.

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

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*, 2024.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving with cached knowledge fusion. *arXiv preprint arXiv:2405.16444*, 2024.

Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu, et al. Kv cache compression, but what must we give in return? a comprehensive benchmark of long context capable approaches. *arXiv preprint arXiv:2407.01527*, 2024.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient llms inference. In *Forty-first International Conference on Machine Learning*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. Mlkv: Multi-layer key-value heads for memory efficient transformer decoding. *arXiv preprint arXiv:2406.09297*, 2024.