LoRC: Low-Rank Compression for LLMs KV Cache with a Progressive Compression Strategy

Anonymous Author(s)
Affiliation
Address
email

Abstract

The Key-Value (KV) cache is a crucial component in serving transformer-based autoregressive large language models (LLMs), enabling faster inference by storing previously computed KV vectors. However, its memory consumption scales linearly with sequence length and batch size, posing a significant bottleneck in LLM deployment. Existing approaches to mitigate this issue include: (1) efficient attention variants integrated in upcycling stages, which requires extensive parameter tuning thus unsuitable to pre-trained LLMs; (2) KV cache compression at test time, primarily through token eviction policies, which often overlook inter-layer dependencies and can be task-specific.

This paper introduces an orthogonal approach to KV cache compression. We propose a *low-rank approximation* of KV weight matrices, allowing for plug-in integration with existing transformer-based LLMs without model retraining. To effectively compress KV cache at the weight level, we adjust for layerwise sensitivity and propose a *progressive compression* strategy, guided by the condition numbers of KV weight matrices. Our method is designed to function without model tuning in upcycling stages or task-specific profiling in test stages. Experiments with 8B to 70B LLaMA models across various tasks show that our approach significantly reduces the GPU memory footprint while maintaining performance.

1 Introduction

Autoregressive large language models (LLMs) such as GPT (Achiam et al., 2023), PaLM (Chowdhery et al., 2023), and LLaMA (Touvron et al., 2023), built upon transformer architectures (Vaswani et al., 2017), have shown remarkable capabilities across a wide range of tasks. However, the attention mechanism underpinning those models poses significant challenges to the efficiency of their deployment, particularly the management of the Key-Value (KV) cache. The KV cache is originally designed to accelerate the generation process by storing intermediate attention vectors, thus avoiding recomputation of shared prefixes for each autoregressively generated token. Despite reducing computational overhead, the KV cache significantly increases memory footprints, as its size scales linearly with both sequence length and batch size.

To address the overhead of the original attention mechanism, one prominent line of work aims to design more efficient attention variants, such as multi-query attention (MQA) (Shazeer, 2019) and group-query attention (GQA) (Ainslie et al., 2023), which inherently reduce the corresponding KV cache. Nevertheless, those techniques typically require upcycling existing vanilla models. Without proper training, their direct application often results in degraded performance (Ribar et al., 2023; Ainslie et al., 2023; Liu et al., 2024b), making them unsuitable for deployment in resource-constrained environments. Recently, Liu et al. (2024a) design a multi-head latent attention (MLA) for efficient inference, utilizing low-rank key-value union compression to reduce KV cache. However, similar to

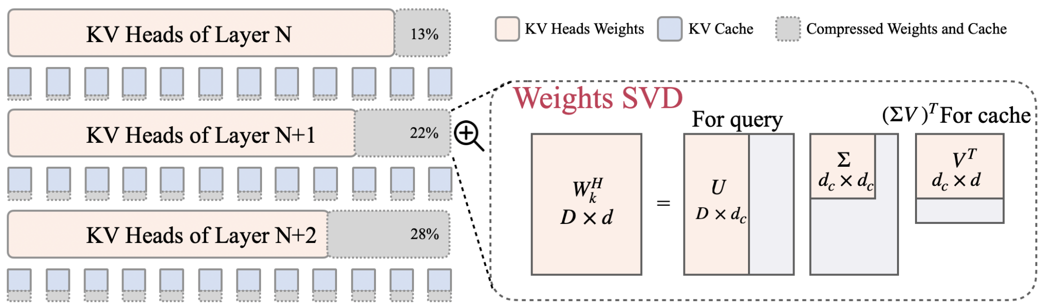

Figure 1: LoRC compresses KV-cache by decomposing the KV weight matrices in attention heads. The progressive compression strategy retains more dimension for KV weights in shallow layers and compresses the KV weights in deep layers more aggressively.

MQA and GQA, MLA is also integrated during the model's training cycle, thus not directly applicable to pre-trained LLMs.

In contrast, another line of work focuses on KV cache compression at test time, primarily achieved by dropping tokens while leaving the backbone model intact. Several works design the token eviction policy based on accumulated attention scores(Sheng et al., 2023; Zhang et al., 2024b; Liu et al., 2024b), or heuristics such as special tokens or and relative distance between tokens (Ge et al., 2023). However, these methods ignore the inter-layer dependency and the resulting eviction policy can be task-specific.

In this paper, we propose to compress KV cache from an orthogonal perspective, *i.e.,* the KV weight matrices. As the KV weight matrices are typically characterized by low-rank properties, we perform a *low-rank approximation* to reduce their dimension and thus compress the resulting KV cache. Recognizing that compressed KV caches inevitably introduce information loss to subsequent layers, and that sensitivity to input changes varies across layers, we introduce a *progressive* compression strategy. This approach is grounded in the calculation of cumulative condition numbers for KV weight matrices across different layers, reflecting their sensitivity and guiding our compression strategy. In this way, we determine layer-specific compression dimensions that effectively balance compression with the preservation of critical information.

Our method is designed for straightforward implementation, requiring neither model profiling nor detailed inspection of the attention structure. It can be directly applied to pre-trained LLMs by extracting weight matrices and leveraging their inherent properties to swiftly determine optimal layer-wise compression. This approach offers a practical and efficient solution for enhancing LLM performance in memory-constrained deployment scenarios, without the need for model retraining or complex eviction strategy composition.

We evaluate our method on 8B, 13B, and 70B LLaMA models that built upon multi-query attention or group-query attention. Experiments across tasks such as commonsense reasoning, reading comprehension, text summarization, and mathematical reasoning, demonstrate that our approach can reduce substantial GPU memory footprint while maintaining minimal impact on performance.

2 Method

We structure this section as follows. In Section 2.1, we detail the process of compressing the KV cache for a single layer using Singular Value Decomposition (SVD) of the weight matrix. Section 2.2 introduces our progressive compression strategy, which determines adaptive compression dimensions for each layer. Figure 1 presents an overview of our method, illustrating the low-rank approximation of the weight matrix and the progressive compression strategy across layers. Because of the space limit, we put related works and preliminaries to Appendix A and B. We also cover consideration of handling various attention mechanisms in Appendix C , and address implementation details specific to the rotary position embedding in Appendix D.

2.1 KV Cache Compression via Low-rank Approximation of Weight Matrices

Unlike previous approaches that focus on token-level eviction strategies or require model retraining, we propose a novel method that operates at the weight matrix level in the attention mechanism. This approach leverages the inherent low-rank properties of these matrices (as shown in Appendix G), allowing for significant compression without the need for complex token selection algorithms or time-consuming model fine-tuning. By applying a low-rank approximation to the weight matrices, we effectively reduce the dimensionality of the KV cache while preserving the essential information flow through the network.

Key Matrix Compression: Figure 1 presents how we implement SVD on the key weight matrices. Specifically, for the i-th head in the MHA attention, we decompose its key matrix $W_k^i \in \mathbb{R}^{D\times d}$ to:

$$\text{SVD}(W_k^i)_{D\times d} = U_{D\times d_c}\Sigma_{d_c\times d_c}V_{d_c\times d}^T = U_{D\times d_c}(\Sigma V^T)_{d_c\times d}. \tag{1}$$

For MHA, there are h attention heads, then the decomposition becomes:

$$\text{SVD}(W_k^H)_{D\times hd} = U_{D\times d_c}(\Sigma V^T)_{d_c\times hd} = U_{D\times d_c}\begin{bmatrix} (A^1)_{d_c\times d} & (A^2)_{d_c\times d} & \cdots & (A^h)_{d_c\times d} \end{bmatrix}, \tag{2}$$

where $(A^i)_{d\times d_c}$ is the i-th block in the matrix $(\Sigma V^T)_{hd\times d_c}$.

Now we have decomposed the key matrix W_k^i to the multiplication of $U_{D\times d_c}$ and $(A^i)_{d_c\times d}$. We will multiply X with $(\Sigma V)_{hd\times d_c}$ as the compressed key, which is stored in the KV cache. For $U_{D\times d_c}$, we incorporate it to the query calculation by updating the original query matrix $W_q^i \in \mathbb{R}^{D\times d}$ to

$$W_{q'}^i = (W_q^i)_{D\times d}(A^i)_{d\times d_c}, \tag{3}$$

then the updated query matrix $W_{q'}^i \in \mathbb{R}^{D\times d_c}$. By compressing the key matrix using SVD, we effectively reduce the size of key cache from $h\times d$ to d_c, where d_c is smaller than hd, reducing the memory footprint while keeping the essential information intact.

Value Matrix Compression: The decomposition for the value matrix follows a similar structure. The only difference lies in that we integrate its left singular vectors to the output matrix. Specifically, the value matrix is decomposed as:

$$\text{SVD}(W_v^H)_{D\times hd} = U_{D\times d_c}(\Sigma V^T)_{d_c\times hd} = U_{D\times d_c}\begin{bmatrix} (B^1)_{d_c\times d} & (B^2)_{d_c\times d} & \cdots & (B^h)_{d_c\times d} \end{bmatrix} \tag{4}$$

where $(B^i)_{d_c\times d}$ is the i-th block in the matrix $(\Sigma V^T)_{d_c\times hd}$. Different from the above operation of key, here we incorporate $U_{D\times d_c}$ to the output matrix, and keep the $(\Sigma V^T)_{d_c\times hd}$ for KV cache.

Consider the output matrix $W_o \in \mathbb{R}^{D\times D}$, we update it by

$$W_{o'} = (U^\top)_{d_c\times D}(W_o)_{D\times D}, \tag{5}$$

then the updated query matrix $W_{o'} \in \mathbb{R}^{d_c\times D}$.

Compression Ratio: The compression strategy effectively reduces the dimensions from $N\times d\times h$ for both keys and values to $N\times d_c$, ensuring data integrity and minimizing overhead. This results in a layer compression ratio $\rho = \frac{d_c}{h\times d}$, which quantifies the extent of the reduction.

2.2 Progressive Compression Strategy

Having established low-rank approximation for compressing weight matrices, we now address its dynamic application across network layers to optimize performance and memory usage. This approach is necessary due to the varying sensitivity of different layers to information loss, which significantly affects overall model efficacy and efficiency.

To tackle this challenge, we propose a *progressive* compression strategy for the KV cache using low-rank approximation of KV weight matrices. Our intuition is that compressed shallow layers could lead to cascading errors that propagate and amplify through the network. Therefore, we analyze the condition numbers of KV weight matrices for each layer to determine *layer-wise* compression dimensions. This approach accounts for each layer's sensitivity to perturbations from previously compressed layers, ensuring output variations remain within acceptable ranges. The progressive nature of our strategy allows for more conservative compression in earlier layers, minimizing the risk of error accumulation throughout the network. By carefully balancing compression across layers, we maintain model integrity while achieving significant memory savings.

Condition Number and Sensitivity Analysis To ensure that the change in the output $\mathbf{b}_l = \mathbf{A}_l\mathbf{x}_l$ remain within a specified range when the input $\mathbf{x}_l$ changes due to compression in previous layers, we need to consider the sensitivity of the output to such changes. Given a weight matrix $\mathbf{A}_l$, its condition number plays a crucial role in determining the allowable change in $\mathbf{x}_l$. The condition number $\kappa(\mathbf{A}_l)$ is defined as:

$$\kappa(\mathbf{A}_l) = |\mathbf{A}_l|_2 \cdot |\mathbf{A}_l^{-1}|_2 = \frac{\sigma_{\max}(\mathbf{A}_l)}{\sigma_{\min}(\mathbf{A}_l)}, \tag{6}$$

where $\sigma_{\max}(\mathbf{A}l)$ and $\sigma_{\min}(\mathbf{A}_l)$ are the largest and smallest singular values of $\mathbf{A}_l$, respectively. To keep the relative change in the output $\mathbf{b}_l$ within a tolerance ϵ, we utilize the standard definition of the condition number to relate it to the allowable relative change in the input $\mathbf{x}_l$:

$$\frac{|\Delta\mathbf{b}_l|_2}{|\mathbf{b}_l|_2} \leq \kappa(\mathbf{A}_l) \cdot \frac{|\Delta\mathbf{x}_l|_2}{|\mathbf{x}_l|_2} \leq \epsilon. \tag{7}$$

Solving for the allowable relative change in $\mathbf{x}_l$, we obtain: $\frac{|\Delta\mathbf{x}_l|_2}{|\mathbf{x}_l|_2} \leq \frac{\epsilon}{\kappa(\mathbf{A}_l)}$. This inequality indicates that the acceptable change in the input $\mathbf{x}_l$ is inversely proportional to the condition number $\kappa(\mathbf{A}_l)$ of the layer's weight matrix. Layers with higher condition numbers are more sensitive to input perturbations, requiring smaller changes in $\mathbf{x}_l$ to maintain the output within the desired range. Given the multi-layer structure of transformers, it's essential to consider not just the condition number of a single layer but the cumulative effect of condition numbers from all preceding layers. This cumulative measure gives a more holistic view of how perturbations might propagate and amplify as data passes through successive layers.

Cumulative Condition Number: To effectively manage this across the network, we calculate the cumulative condition number as a estimated layerwise sensitivity and then derive the compression dimension. For a model with L layers, we calculate the cumulative condition number for each layer l by multiplying the condition numbers of the subsequent layers:

$$\tilde{\kappa}_l = \prod_{i=l}^{L} \kappa(W_k^i) \cdot \kappa(W_v^i). \tag{8}$$

This cumulative condition number $\tilde{\kappa}_l$ reflects the total amplification of input perturbations from layer l to the output layer.

Compression Dimension: Based on the cumulative condition number, we then adjust the compression dimensions for each layer to balance the fidelity and compression rate. More sensitive layers (those with higher cumulative condition numbers) will have less aggressive compression to preserve critical information, whereas layers with lower sensitivity can be compressed more substantially without significantly affecting the overall network performance. We compute the compressed dimension d_l for each layer by scaling $\tilde{\kappa}_l$ using the following function:

$$d_l = d_{\max} \times \left[1 - \left(\frac{\max_{i\in[1:L]} \log(\tilde{\kappa}_i) - \log(\tilde{\kappa}_l)}{\max_{i\in[1:L]} \log(\tilde{\kappa}_i) - \min_{i\in[1:L]} \log(\tilde{\kappa}_i)}\right) \times \left(1 - \frac{d_{\min}}{d_{\max}}\right)\right], \tag{9}$$

where $d_{\max}$ is the maximum allowable compressed dimension, and $d_{\min}$ is the minimum one. The logarithmic scale mitigates the effect of large variations in the cumulative condition numbers, providing a more balanced sensitivity metric across layers. This equation ensures that layers with higher sensitivity (larger $\tilde{\kappa}_l$) retain more dimensions (larger d_l), while less sensitive layers can be compressed more aggressively.

3 Experiment

3.1 Models

We conduct experiments using two attention mechanisms, Multi-Head Attention (MHA) (Vaswani et al., 2017) and Graph Query Attention (GQA) (Ainslie et al., 2023), across three models: LLaMA-2-13B, LLaMA-3-Instruct-8B, and LLaMA-3-Instruct-70B. The LLaMA-2 family incorporates the MHA mechanism, while the LLaMA-3 family is based on the GQA framework. We list the model specifications in Table 3. Note that for the models based on MHA, the number of KV heads is equal to the number of attention heads, so the weight matrices of KV are square matrices. The models based on GQA use an intermediate number of key-value heads to group the query heads, with an adjustment on the shape of KV weight matrices. Due to space limit, we introduce datasets and implementation details in Appendix F.

3.2 Main Results

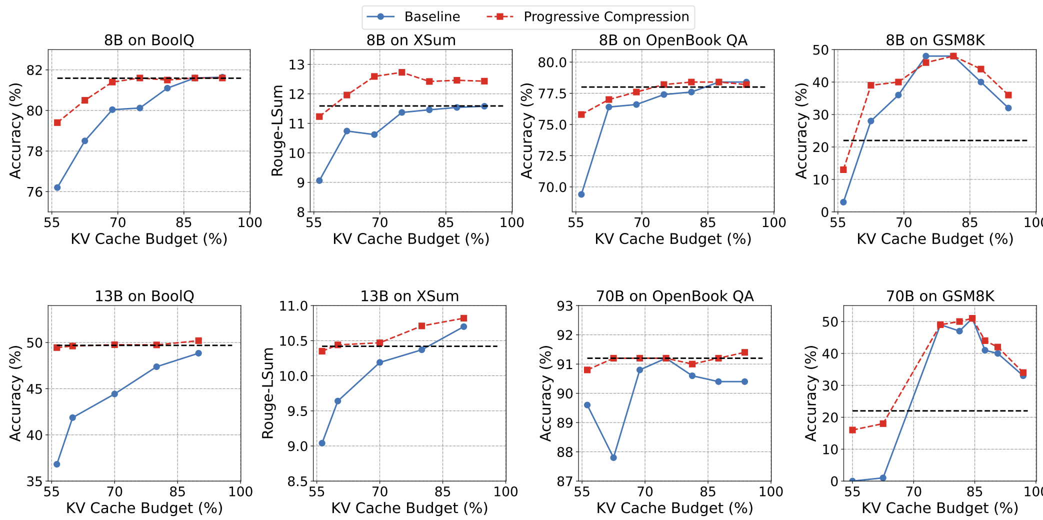

Figure 2: Performance of KV cache compression on LLaMA models. LoRC compresses the KV weights with a progressive strategy, while the baselines compress each layer with the same ratio. The horizontal dashed line indicates the performance with a full-cache model.

Figure 2 presents our main results on four datasets with different KV cache budgets. Compared to the full-cache model, LoRC achieves on-par performance with a significant compression ratio, and the performance degradation is still nearly negligible with a 60% compression ratio on most datasets. When slightly compressed, LoRC could even enhance model performance in some cases. Note that our method requires no model training or model profiling, the only efforts are SVD on weight matrices which requires minimal computational cost compared to the LLM inference. Such plug-and-play merits make our method easily integrable in resource-constrained environments, enabling efficient model deployment with limited KV cache budgets.

In Figure 2, one interesting observation is that in some cases the model with a compressed KV cache leads to better performance. Particularly, on the GSM8K dataset, performing KV cache compression leads to more than 10% performance improvement. This phenomenon aligns with findings reported in the literature (Ge et al., 2023). Also, similar effects have been documented in the context of improving reasoning by applying low-rank decomposition on the MLP layers (Sharma et al., 2023). We believe this phenomenon demonstrates the feasibility of conducting task-specific profiling for better performance, or adapting our proposed method in model finetuning.

3.3 Single Layer Profiling

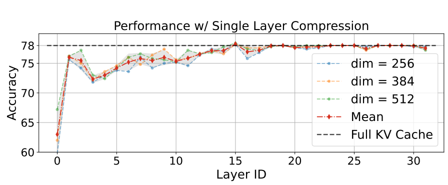

Figure 3: Single layer profiling results with LLaMA-3-Instruct-8B on OpenBook QA.

To investigate the impact of compression at different layers, we conduct experiments on single-layer compression as shown in Fig. 3. We use LLaMA-3-Instruct-8B on OpenBook QA for this experiment. The original dimension of the KV head is 1024, and we select compression dimensions from $[256, 384, 512]$ to compress each single layer while keeping all other layers untouched.

Figure 3 shows clear layer-specific variability, indicating that some layers are more susceptible to compression than others, particularly in the shallow layers. It is observed that the deep layers (*i.e.,* layers 15–31 of the 32-layer LLaMA-3-Instruct 8B model), despite the reduction in dimensions, maintain performance closely approaching the full KV Cache baseline. This suggests that these layers can sustain robust performance even when subjected to significant parameter reduction. This finding supports our progressive compression strategy for optimizing model efficiency without significantly compromising the model's effectiveness.

3.4 Curse of Shallow Layers

Table 1: Performance comparison between our method and compression on different layers on OpenBookQA. For our progressive compression strategy, we report the performance at the 60% overall compression ratio. For layer-0 compression and shallow blocks compression, we use a 50% layer compression ratio within the chosen strategy. Hence, the overall compression ratio is 98.44% for the layer-0 compression, and 93.75% for the shallow blocks compression.

Model	Baseline	Ours	Layer 0	Shallow Blocks (1/8)
LLaMA-2-13b	76.6	77.4 (↑ 0.8)	77.2 (↑ 0.6)	74.8 (↓ 1.8)
LLaMA-3-Instruct-8b	78.0	77.4 (↓ 0.6)	67.2 (↓ 10.8)	61.4 (↓ 16.6)
LLaMA-3-Instruct-70b	91.2	91.2 (↑ 0.0)	84.2 (↓ 7.0)	23.2 (↓ 68.0)

To validate the intuition of the progressive compression strategy that the noise caused by shallow compressed layers will be amplified more after propagation, we compare it to compressing the first layer and the shallow blocks (i.e., the first 1/8 layers in a model) on 3 LLaMA models.

Table 1 shows how the compressed shallow layers impact the model performance, taking the baseline full-cache model and our method as reference. The results indicate that compressing only the first layer can lead to a performance decline, with reductions ranging from minimal to moderate. For instance, the LLaMA-3-70B gives a 7.0% decrease, while the LLaMA-3-Instruct-8b shows a more substantial drop of 10.8%. When compressing the shallow blocks, the impact is more pronounced, highlighting a significant sensitivity to shallow layer compression. These findings underscore the importance of careful layer selection in compression strategies and validate the effectiveness of our progressive compression method, as the choice of layer to compress can have a substantial impact on model performance, particularly in larger or more complex models.

3.5 Memory Footprint Reduction Analysis

Table 2: Summary of Model Sizes, KV cache usage and performance drop. Experiments were conducted with a batch size of 64 and a sequence length of 2048 for all models.

Model	KV Cache					Average Performance Drop
	Full	dim	dim_c	Ours	Compression Ratio	
LLaMA-2-13B	50G	5120	2048	27.5G	55%	0.47%
LLaMA-3-8B	8G	1024	512	4.8G	60%	0.92%
LLaMA-3-70B	20G	1024	512	11G	55%	0.22%

We report the memory footprint reduction in Table 2. By controlling the performance drop averaged on the four tasks less than 1%, we can achieve a considerable compression ratio of 55%-60%. For the LLaMA-3 models in which the GQA has already been employed to save the KV cache, we further achieve a significant compression ratio. Note that we have excluded the GSM8k results for the performance drop calculation for a fair comparison.

4 Conclusions

In conclusion, we proposed LoRC, a novel approach to KV cache compression that capitalizes on the inherent low-rank properties of weight matrices. Our method employs a progressive layer-wise compression strategy, implementing a post-hoc low-rank approximation to circumvent the complexities and limitations associated with token-level eviction strategies and model retraining. This universally applicable solution preserves model integrity and performance across diverse tasks, attention mechanisms, and model scales. Our comprehensive experimental results demonstrate that LoRC significantly reduces GPU memory requirements while minimally impacting performance. This approach offers a robust and efficient solution to the challenge of KV cache compression, paving the way for more resource-efficient deployment of large language models.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

de Jong, M., Zemlyanskiy, Y., Ainslie, J., FitzGerald, N., Sanghai, S., Sha, F., and Cohen, W. Fido: Fusion-in-decoder optimized for stronger performance and faster inference. *arXiv preprint arXiv:2212.08153*, 2022.

Ge, S., Zhang, Y., Liu, L., Zhang, M., Han, J., and Gao, J. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Guan, Y., Li, Z., Leng, J., Lin, Z., and Guo, M. Transkimmer: Transformer learns to layer-wise skim. *arXiv preprint arXiv:2205.07324*, 2022.

Holmes, C., Tanaka, M., Wyatt, M., Awan, A. A., Rasley, J., Rajbhandari, S., Aminabadi, R. Y., Qin, H., Bakhtiari, A., Kurilenko, L., et al. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.

Izacard, G. and Grave, E. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*, 2020.

Lin, C.-Y. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W04-1013`.

Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024b.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Mu, J., Li, X., and Goodman, N. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36, 2024.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *ArXiv*, abs/1808.08745, 2018.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Heek, J., Xiao, K., Agrawal, S., and Dean, J. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5, 2023.

Ribar, L., Chelombiev, I., Hudlass-Galley, L., Blake, C., Luschi, C., and Orr, D. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.

Sharma, P., Ash, J. T., and Misra, D. The truth is in there: Improving reasoning in language models with layer-selective rank reduction. *ArXiv*, abs/2312.13558, 2023.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Sun, T., Liu, X., Zhu, W., Geng, Z., Wu, L., He, Y., Ni, Y., Xie, G., Huang, X., and Qiu, X. A simple hash-based early exiting approach for language understanding and generation. *arXiv preprint arXiv:2203.01670*, 2022.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yu, H., Yang, Z., Li, S., Li, Y., and Wu, J. Effectively compress kv heads for llm. *arXiv preprint arXiv:2406.07056*, 2024.

Zhang, Y., Gao, B., Liu, T., Lu, K., Xiong, W., Dong, Y., Chang, B., Hu, J., Xiao, W., et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024a.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36, 2024b.

Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., and Wei, F. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.

A Related Works

A.1 Attention Mechanism

Attention mechanisms in Transformer models have evolved to enhance efficiency and effectiveness (Vaswani et al., 2017). Multi-Query Attention (MQA) (Shazeer, 2019) reduces memory requirements during decoding, while Grouped-Query Attention (GQA) (Ainslie et al., 2023) balances efficiency and performance by sharing key and value heads among query groups. Recently, Liu et al. (2024a) introduced Multi-head Latent Attention (MLA), using low-rank key-value union compression to optimize inference. However, these approaches are typically integrated during model training, limiting their applicability to pre-trained LLMs. Parallel research efforts have targeted inference efficiency improvements. For example, Pope et al. (2023) developed multi-dimensional partitioning techniques, and de Jong et al. (2022) optimized the Fusion-in-Decoder (FiD) approach (Izacard & Grave, 2020) for more efficient inference. Holmes et al. (2024) introduces SplitFuse which leverages dynamic prompt and generation decomposition and unification to further improve continuous batching and system throughput. In this paper, we contribute to this line of research by improving inference efficiency through the compression of KV cache. Our approach leverages the low-rank property of the attention weight matrices, offering a plug-and-play method to reduce the memory footprint of LLMs during inference without requiring model retraining.

A.2 KV Cache Compression

As Large Language Models (LLMs) continue to grow in size and complexity, efficient management of their memory usage during inference has become a critical challenge. Early efforts to compress token hidden states (Guan et al., 2022; Sun et al., 2022; Zhou et al., 2020) are limited to non-autoregressive models and require retraining, thus motivating research into pruning tokens in the KV cache of auto-regressive LLMs. For instance, Mu et al. (2024) learns to compress prompts into a few special tokens to reduce memory pressure during caching, but this token prediction requires model retraining and could be an expensive overhead during inference. Several methods design token eviction policies based on accumulated attention scores (Sheng et al., 2023; Zhang et al., 2024b; Liu et al., 2024b), or heuristics such as special tokens and relative distance between tokens (Ge et al., 2023). However, these approaches often overlook inter-layer dependencies, potentially resulting in task-specific eviction policies that may not generalize well across different applications. In contrast to token-dropping methods, our study takes a different tack. We focus on compressing the KV cache from the perspective of weight matrix dimension reduction. Importantly, our progressive compression strategy carefully addresses the issue of error propagation across compressed layers, a consideration often ignored in previous methods.

A few studies have explored customized cache budgets across different layers in the context of token dropping, yet no definitive consensus has been reached on the most effective strategies. Zhang et al. (2024a) suggest increasing compression intensity in higher layers based on the assumption that these layers contain less critical information. Conversely, Liu et al. (2024b) argue that significant tokens exhibit greater variability at higher layers, thus larger caches are required to reduce cache misses. While these approaches demonstrate understanding of layer-specific requirements, they depend heavily on task-specific attention patterns. Our approach diverges fundamentally by adopting an orthogonal perspective to compression, focusing on weight matrix dimension reduction rather than token eviction. This approach enables us to establish error propagation bounds across the network and to guide our progressive compression strategy effectively. It eliminates the need to analyze attention patterns for eviction policy design, simplifying implementation and enhancing general applicability across different LLMs.

Concurrently, Liu et al. (2024a) and Yu et al. (2024) modify attention mechanisms to manage KV caches more efficiently during inference. While these methods align with our philosophy of altering attention dynamics, they require either pretraining adjustments or extensive model finetuning to accommodate the modified attention schemas, limiting their practicality in deployed systems. In contrast, our method requires no such training or fine-tuning, offering a plug-and-play solution that seamlessly integrates with pre-trained models to deliver efficient compression without compromising the model's integrity or performance.

B Preliminary: Attention Mechanism and KV Cache

Transformer-based language models use self-attention to weigh the importance of different tokens, thus allowing for the model to focus on different parts of the input sequence. Given an input $X \in \mathbb{R}^{N \times D}$, where N is the sequence length and D is the dimensionality of each token's embedding, we compute the Query (Q), Key (K), and Value (V) matrices by multiplying X with their respective weight matrices: $Q = XW_q, K = XW_k, V = XW_v$.

Then the attention mechanism is as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{10}$$

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}^1, \ldots, \text{head}^h)W_o, \tag{11}$$

where

$$\text{head}^i = \text{Attention}(XW_q^i, XW_k^i, XW_v^i). \tag{12}$$

Here, W_q^i, W_k^i, and W_v^i are the weight matrices for the i-th attention head, and W_o is the weight matrix for the output linear transformation.

In autoregressive transformers, the computation of attention scales quadratically (*i.e.,* $\mathcal{O}(N^2)$) with the sequence length N, as every token in the sequence computes interactions with every other token. Such scaling is impractical for very large inputs or real-time applications, where speed and efficiency are crucial.

To address this computational bottleneck, KV caches store the results of previous computations of the Key/Value matrices. When processing subsequent tokens, the model can retrieve Keys and Values from the cache rather than recomputing them, thereby reducing the number of operations to a linear scale with respect to the sequence length. This method trades off increased memory usage for a reduction in computational overhead. The size of KV cache per layer is defined as below:

$$C_{k,v} = b \times N \times n \times d, \tag{13}$$

where b is the batch size, N is the max sequence length in the batch, n is the number of K/V head and d is the head dimension. As seen above, the memory footprint cost of caching can be substantial because the KV cache scales linearly with both sequence length and batch size. This drives the need of compression methods to reduce KV cache for LLM deployment in resource-constrained scenarios.

C Multi-head Attention and Group-query Attention

The above derivation in Section 2.1 holds for standard MHA, where the model dimension D equals to the multiplication of number of head and head dimension $h \times d$. For GQA, the number of KV heads is reduced as shown in Table 3. To adapt such implementation, we can still follow the above procedure for cache compression. After fetching the Key/Value from cache, we just need to repeat them according to the number of the total attention heads.

D Adjusted Position Embedding

Su et al. (2024) propose a rotary position embedding (RoPE) and it has been used in most recent LLMs. Applying RoPE to self-attention gives

$$q_m^T k_n = (R_{\Theta,m}^d W_q x_m)^T (R_{\Theta,n}^d W_k x_n) = x^T W_q^T R_{\Theta,n-m}^d W_k x_n, \tag{14}$$

where Θ is a pre-defined rotary matrix, m and n denotes the token position. The $R_{\Theta,n-m}^d$ can be decomposed as $(R_{\Theta,m}^d)^T R_{\Theta,n}^d$ to rotate the query and key, respectively. In the original KV cache stores rotated key, thus we need to adjust this position embedding pipeline to ensure the compressed keys are compatible with the rotary operation. Specifically, we only keep $X(\Sigma V)_{D \times d_c}^\top$ in cache, and incorporate the other components into the calculation of the query to streamline the process.

E Error Bounds for KV Cache Compression

In this section, we derive error bounds for our KV cache compression method, considering both individual layer errors and their propagation through a deep network. These theoretical results provide insights into how compression affects the network's performance and guide the selection of compression levels to balance efficiency and accuracy.

E.1 Single Layer Error Bound

Theorem 1 *Let $W \in \mathbb{R}^{m\times n}$ be a weight matrix (either key or value), and let $W_k \in \mathbb{R}^{m\times n}$ be its rank-k approximation obtained via truncated singular value decomposition (SVD). For any input vector $x \in \mathbb{R}^n$, the error introduced by the approximation is bounded by:*

$$\|Wx - W_k x\|_2 \leq \sigma_{k+1}\|x\|_2, \tag{15}$$

where σ_{k+1} is the $(k+1)$-th singular value of W.

Proof.

Let $W = U\Sigma V^\top$ be the full SVD of W, where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices, and $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$ with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$.

The rank-k approximation W_k is given by:

$$W_k = U_k \Sigma_k V_k^\top,$$

where U_k, Σ_k, and V_k are truncated versions of U, Σ, and V, respectively, keeping only the first k singular values and corresponding vectors.

We have:

$$\begin{aligned}
\|Wx - W_k x\|_2 &= \|(W - W_k)x\|_2 \\
&= \|U(\Sigma - \Sigma_k)V^\top x\|_2 \\
&= \|(\Sigma - \Sigma_k)V^\top x\|_2, \quad \text{since } U \text{ is orthogonal} \\
&= \|\operatorname{diag}(0, \ldots, 0, \sigma_{k+1}, \ldots, \sigma_n)V^\top x\|_2 \\
&\leq \sigma_{k+1}\|V^\top x\|_2 \\
&= \sigma_{k+1}\|x\|_2, \quad \text{since } V \text{ is orthogonal.}
\end{aligned}$$

□

E.2 Error Propagation Bound

Theorem 2 *Consider an L-layer network where each layer i applies a linear transformation W_i followed by a nonlinearity ϕ with Lipschitz constant L_ϕ. Let $\tilde{W}_i$ be the compressed version of W_i obtained via truncated SVD with rank k_i. The error at the output of the network is bounded by:*

$$\|x_L - \tilde{x}_L\|_2 \leq \sum_{i=1}^{L}\left(\sigma_{k_i+1}^{(i)} L_\phi^{L-i} \prod_{j=i+1}^{L}\|W_j\|_2\right), \tag{16}$$

where x_L and $\tilde{x}_L$ are the outputs of the original and compressed networks, respectively; $\sigma_{k_i+1}^{(i)}$ is the (k_i+1)-th singular value of W_i; $\|W_j\|_2$ denotes the spectral norm of W_j; and L_ϕ is the Lipschitz constant of the activation function ϕ.

Proof.

Let x_i and $\tilde{x}_i$ denote the outputs of the i-th layer in the original and compressed networks, respectively. We prove by induction:

Base Case ($i = 1$).

Using Theorem 1 and the Lipschitz property of ϕ:

$$\begin{aligned}\|x_1 - \tilde{x}_1\|_2 &= \|\phi(W_1 x_0) - \phi(\tilde{W}_1 x_0)\|_2 \\ &\leq L_\phi \|W_1 x_0 - \tilde{W}_1 x_0\|_2 \\ &\leq L_\phi \sigma_{k_1+1}^{(1)} \|x_0\|_2.\end{aligned}$$

Inductive Step.

Assume the inductive bound holds for layer $i-1$. For layer i:

$$\begin{aligned}\|x_i - \tilde{x}_i\|_2 &= \|\phi(W_i x_{i-1}) - \phi(\tilde{W}_i \tilde{x}_{i-1})\|_2 \\ &\leq L_\phi \|W_i x_{i-1} - \tilde{W}_i \tilde{x}_{i-1}\|_2 \\ &\leq L_\phi \left(\|W_i (x_{i-1} - \tilde{x}_{i-1})\|_2 + \|(W_i - \tilde{W}_i)\tilde{x}_{i-1}\|_2 \right) \\ &\leq L_\phi \left(\|W_i\|_2 \|x_{i-1} - \tilde{x}_{i-1}\|_2 + \sigma_{k_i+1}^{(i)} \|\tilde{x}_{i-1}\|_2 \right).\end{aligned}$$

We can bound $\|\tilde{x}_{i-1}\|_2$ using the triangle inequality:

$$\|\tilde{x}_{i-1}\|_2 \leq \|x_{i-1}\|_2 + \|x_{i-1} - \tilde{x}_{i-1}\|_2.$$

Assuming that $\|x_{i-1}\|_2$ is bounded (which is reasonable in practice due to normalization techniques), and applying the inductive hypothesis, we can express $\|x_i - \tilde{x}_i\|_2$ in terms of the accumulated errors up to layer i.

By recursively applying this inequality and summing over all layers, we obtain the bound stated in Theorem 2.

$\square$

E.2.1 Adjustment for Non-1-Lipschitz Activation Functions

For activation functions where $L_\phi > 1$, such as SwiGLU and GELU used in LLaMA models, the error bound adjusts to account for the increased potential for error amplification.

Modified Error Bound:

$$\|x_L - \tilde{x}_L\|_2 \leq \sum_{i=1}^{L} \left(\sigma_{k_i+1}^{(i)} L_\phi^{L-i} \prod_{j=i+1}^{L} \|W_j\|_2 \right). \tag{17}$$

Explanation:

In this adjusted bound, L_ϕ^{L-i} reflects the cumulative effect of the activation functions' Lipschitz constant over the remaining layers of the network. A larger L_ϕ implies that errors can grow more significantly as they propagate, highlighting the importance of carefully choosing compression levels in networks with such activation functions.

E.3 Bound on Performance Degradation

Corollary 2.1 *Let $f(x)$ be the output of the original network and $\tilde{f}(x)$ be the output of the compressed network for input x. Assume the loss function $\mathcal{L}$ is $L_{\mathcal{L}}$-Lipschitz in its first argument. The performance degradation is bounded by:*

$$|\mathcal{L}(f(x), y) - \mathcal{L}(\tilde{f}(x), y)| \leq L_{\mathcal{L}} \|x_L - \tilde{x}_L\|_2 \leq L_{\mathcal{L}} \sum_{i=1}^{L} \left(\sigma_{k_i+1}^{(i)} L_\phi^{L-i} \prod_{j=i+1}^{L} \|W_j\|_2 \right), \tag{18}$$

where y is the true label.

Proof.

Since $\mathcal{L}$ is $L_{\mathcal{L}}$-Lipschitz in its first argument:

$$|\mathcal{L}(f(x), y) - \mathcal{L}(\tilde{f}(x), y)| \leq L_{\mathcal{L}} \|f(x) - f(\tilde{x})\|_2 = L_{\mathcal{L}} \|x_L - \tilde{x}_L\|_2.$$

Substituting the bound from Theorem 2, we obtain:

$$|\mathcal{L}(f(x), y) - \mathcal{L}(f(\tilde{x}), y)| \leq L_{\mathcal{L}} \sum_{i=1}^{L} \left(\sigma_{k_i+1}^{(i)} L_\phi^{L-i} \prod_{j=i+1}^{L} \|W_j\|_2 \right).$$

□

This corollary connects the theoretical error bounds to practical performance metrics. It shows that the degradation in the loss function due to compression is directly bounded by the cumulative error at the network's output. Since the loss function is $L_{\mathcal{L}}$-Lipschitz, a bounded change in the output leads to a bounded change in the loss.

Table 3: Model Architectures.

Model	Attention	Layers	Heads	KV Heads	Head Dimension	Model Dimension	Weight Shape
LLaMA-2-13B	MHA	40	40	40	128	5120	5120×5120
LLaMA-3-Instruct-8B	GQA	32	32	8	128	4096	4096×1024
LLaMA-3-Instruct-70B	GQA	80	64	8	128	8192	8192×1024

F Experiment Settings

F.1 Dataset

We follow Touvron et al. (2023) to evaluate our methods on the following tasks: **BoolQ** (Clark et al., 2019) for reading comprehension, **XSum** (Narayan et al., 2018) for text summarization. **Openbook QA** (Mihaylov et al., 2018) for commonsense reasoning, and **GSM8K** (Cobbe et al., 2021) for mathematical reasoning. We use ROUGE score (Lin, 2004) as the evaluation metric for XSum and accuracy for the other tasks. We report 2-shot results for LLaMA-2 models on BoolQ, and 0-shot results for other settings.

F.2 Implementation Details

In practice, we set thresholds to exclude compression on layers with high cumulative condition numbers: 30 for LLaMA-3-Instruct-8B, and 90 for LLaMA-2-13B and LLaMA-3-Instruct-70B. The d_{max} equals to the original head dimension, while d_{min} varies based on the target compression ratio. For baseline methods, we have the same refrained layers while applying the uniform compression ratios across compressed layers instead of using a progressive compression strategy.

G Reconstruction Error of Matrix SVD

In our approach, we conduct layer-wise weight matrix decomposition and reconstruction. In this section, we show that these matrices are low-rank and therefore can be reconstructed with low-dimension matrices, resulting in negligible reconstruction error. This suggests that instead of designing complex eviction policies at the token level, we can focus on the weight matrix level

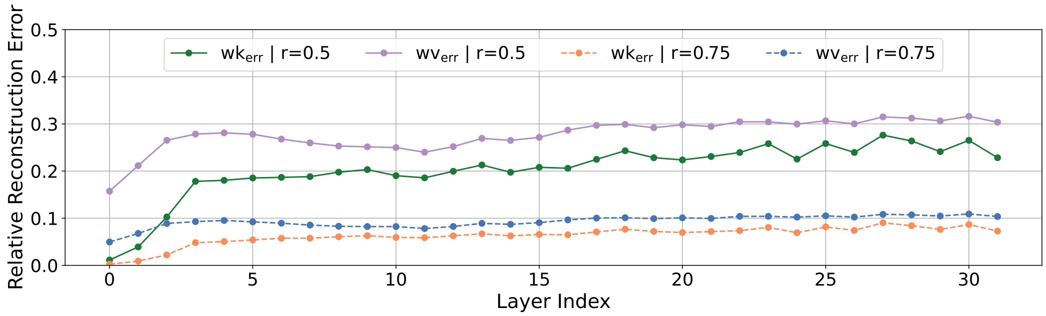

Figure 4: Layerwise relative reconstruction errors. wk_{err} and wv_{err} denote the relative difference between the original key/value matrices and their corresponding low-rank approximations measured using the Frobinus norm. The compression ratio is computed as $r = \frac{d_c}{N_h \times d_h}$, where N_h is the number of attention heads and d_h, d_c is the original and compressed hidden dimensions respectively.

to develop a KV cache compression method. This approach eliminates the need to scrutinize attention patterns to determine which tokens should be dropped. We present the relative reconstruction error in Figure 4, which is computed using the Frobenius norm. For a matrix M, the Frobenius norm is defined as:

$$\|M\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |m_{ij}|^2}. \tag{19}$$

The relative reconstruction error ε is calculated as:

$$\varepsilon = \frac{\|M - \hat{M}\|_F}{\|M\|_F} \tag{20}$$

where M is the original matrix and $\hat{M}$ is the reconstructed matrix obtained through truncated SVD.

This approach enables us to quantify the accuracy of our low-rank approximation for each matrix. It is important to note that although Figure 4 demonstrates that reconstruction errors are similar across all layers, with shallow layers exhibiting even lower errors, this does not imply that we can directly compress shallow layers aggressively or compress all layers uniformly. In fact, compression errors propagate and amplify throughout the network as we illustrated in Section 2.2. To this end, we propose the progressive compression strategy and it is theoretically and empirically effective in minimizing the overall error accumulation.

H Experiment Details

For all experiments except those involving the LLaMA-3-70B model, we utilize a single node equipped with 4 A100 GPUs. For the LLaMA-3-70B model, we employ a node with 8 V100 GPUs.

I Implementation Time for SVD

The calculation of SVD is efficient based on the Numpy library. For LLaMA-3-Instruct-70B, the largest model used in our experiments, the all-layer (80 layers in total) SVD takes only 40 seconds.

J Block Compression

Figure 5 examines the impact of block-wise compression. Having investigated the effects of single-layer and shallow-layer compression, we now extend our exploration to such a more granular approach,

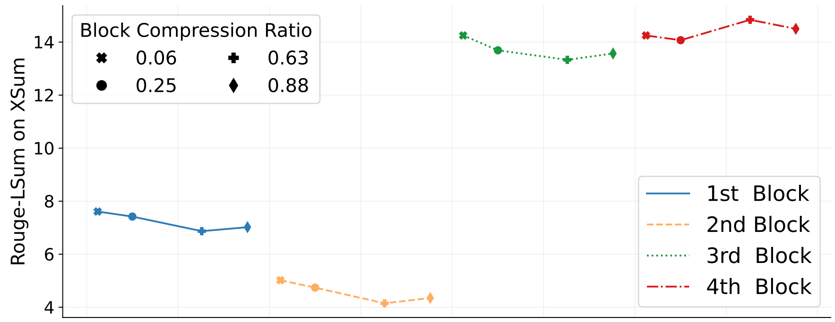

Figure 5: Block compression on XSum with LLaMA-3-Instruct-70B.

which allows for a targeted analysis of grouped compressed layers within the models. We use the 80-layer LLaMA-3-Instruct-70B model in this experiment. It is divided into four 20-layer blocks, and we implement compression ratios ranging from 0.06 to 0.88 in each block. When compressing one block, the other blocks stay untouched. The results reveal that the 4th and 3rd blocks exhibit higher resilience to compression, compared to the 1st and 2nd blocks which show more significant declines in performance. This suggests that deeper blocks in this model architecture may inherently possess higher redundancy or are less critical to the model's overall performance, indicating potential areas for efficiency improvements without substantial loss in output quality.