More Tokens, Lower Precision: Towards the Optimal Token-Precision Trade-off in KV Cache Compression

Anonymous ACL submission

Abstract

As large language models (LLMs) process increasing context windows, the memory usage of KV cache has become a critical bottleneck during inference. The mainstream KV compression methods, including KV pruning and KV quantization, primarily focus on either token or precision dimension and seldom explore the efficiency of their combination. In this paper, we comprehensively investigate the token-precision trade-off in KV cache compression. Experiments demonstrate that storing more tokens in the KV cache with lower precision, i.e., quantized pruning, can significantly enhance the long-context performance of LLMs. Furthermore, in-depth analysis regarding token-precision trade-off from a series of key aspects exhibit that, quantized pruning achieves substantial improvements in retrieval-related tasks and consistently performs well across varying input lengths. Moreover, quantized pruning demonstrates notable stability across different KV pruning methods, quantization strategies, and model scales. These findings provide valuable insights into the token-precision trade-off in KV cache compression. We plan to release our code in the near future.

1 Introduction

As large language models (LLMs) have been widely used in various scenarios, such as document summarization (Zou et al., 2024), code completion (Roziere et al., 2023) and agent framework (Shridhar et al., 2020), there is a growing demand for models with larger context windows to handle more extensive inputs. As a result, GPT-4 (Achiam et al., 2023) have been extended to support 200k input tokens, and Gemini 1.5 (Reid et al., 2024) to 10M tokens. However, these powerful long-context capabilities come at the expense of significantly increased memory storage for the cached key and value (KV) states (Waddington et al., 2013). Take Llama3-8B (Dubey et al., 2024) as an example, storing the KV cache of a single message with 100k tokens requires a high memory overhead of 20GB. Furthermore, as the decoding process heavily relies on the GPU memory bandwidth, the extensive KV cache also leads to dramatically increased decoding time (Fu, 2024).

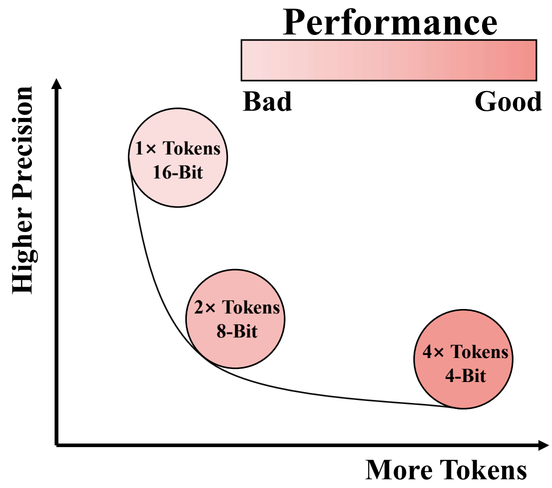

Figure 1: More tokens, lower precision leads to better performance.

To efficiently serve LLMs, various approaches have been proposed to compress KV cache (Pope et al., 2023). The predominant methods involve compressing the KV cache along two primary dimensions: token or precision. For the token dimension, KV pruning (or KV eviction) methods eliminate unimportant tokens to maintain a fixed KV cache size (Xiao et al., 2023; Zhang et al., 2024b; Ren and Zhu, 2024; Li et al., 2024). For the precision dimension, KV quantization technique reduces memory usage by approximating KV cache with lower precisions, like 8-bit or even lower (Sheng et al., 2023; Liu et al., 2024c; Hooper et al., 2024; Yang et al., 2024b). However, these existing works focuses only one dimension, either token or precision, leaving the trade-off between these two orthogonal factors largely under-explored.

In this paper, we comprehensively investigate the token-precision trade-off in KV cache compression. First, we examine the feasibility of combining KV pruning and quantization under fixed budget. We demonstrate that ***storing more tokens in the KV cache with lower precision***, which we call ***quantized pruning***, can significantly enhance the long-context performance of LLMs. For example, with the same KV cache budget, storing $4\times$ tokens in 4-bit precision outperforms storing $1\times$ tokens in 16-bit precision across various downstream long-context tasks with various input length, as shown in Figure 1. Moreover, in extremely low-resource scenarios, quantized pruning effectively preserves performance, whereas relying solely on KV pruning or quantization often leads to a significant performance collapse.

Furthermore, we conduct in-depth analysis regarding token-precision trade-off from series of key aspects, including the impact on various downstream task types and input lengths, model scaling effect, ablation on quantization strategies and fine-grained exploration of layer-wise quantized pruning. From the extensive experiments, we observe that quantized pruning achieves substantial improvements in retrieval-related tasks and consistently performs well across varying input lengths. Moreover, quantized pruning demonstrates strong feasibility across different KV pruning methods, quantization strategies, and model scales, showcasing notable stability. The analysis on token-precision trade-off presents a more substantial compression potential compared to compressing along a single dimension. We believe that our findings could offer valuable insights for developing more effective KV compression strategies in future research.

2 Related Work

KV Pruning KV pruning compresses the KV cache along the token dimension by selectively removing unimportant tokens to reduce memory usage. Mainstream methods typically identify important tokens based on attention scores, as seen in (Liu et al., 2024b; Zhang et al., 2024b; Oren et al., 2024; Li et al., 2024). Other methods use alternative factors such as initial tokens (Xiao et al., 2023), variance (Ren and Zhu, 2024), special tokens (Ge et al., 2024) or the L2 norm (Devoto et al., 2024) to determine token importance. Recent studies delve deeper into optimizing the allocation of KV cache memory budgets. Some explore KV cache budget allocation strategies across layers (Cai et al., 2024; Yang et al., 2024a), while other studies explore head-level KV cache budget allocation (Feng et al., 2024; Tang et al., 2024; Fu et al., 2024; Xiao et al., 2024).

KV Quantization KV quantization compress KV cache from the precision dimension by storing KV cache using a reduced number of bits. FlexGen (Sheng et al., 2023) utilizes group-wise 4-bit quantization for both key and value cache. KIVI (Liu et al., 2024c) applies per-channel quantization on key cache and per-token quantization on value cache. KVQuant (Hooper et al., 2024) and CQ (Zhang et al., 2024a) use RoPE-related quantization, while KVQuant also preverses outliers without quantization. Atom (Zhao et al., 2024) reorders the outlier channels for fine-grained group quantization with mixed-precision. GEAR (Kang et al., 2024) uses low-rank approximation for quantization. QAQ (Dong et al., 2024) and MiKV (Yang et al., 2024b), inspired by the KV pruning methods, store discarded tokens using lower bit precision while retaining important tokens in full precision.

Other KV Compression Methods Compressing KV cache from other dimensions typically requires modifying the model architecture, which usually necessitates additional training for adaptation. For the layer dimension, LCKV (Wu and Tu, 2024), CLA (Brandon et al., 2024) and MLKV (Zuhri et al., 2024) reduce memory usage by sharing the KV cache across adjacent layers. ShortGPT (Men et al., 2024) and DynamicSlicing (Dumitru et al., 2024) achieve compression by eliminating redundant layers. YOCO (Sun et al., 2024) changes the model structure and shares a single global KV cache across layers. For the head dimension, MQA (Shazeer, 2019) and GQA (Ainslie et al., 2023) share the KV cache within each head groups. DeepSeek-v2 (Liu et al., 2024a) employs dimension-reduction techniques to compress all heads into a single low-rank vector. These lines of work is orthogonal to ours, as they can also be combined together.

3 Preliminaries

The decoder-only transformer model consists of a stack of transformer decoder blocks, each comprising two main components: self-attention module and the feed-forward network (FFN) module. During inference, KV cache is implemented within the self-attention module and operates in two distinct phases: i) the prefill phase, where the input prompt is used to generate KV cache for each transformer layer of LLMs; and ii) the decoding phase, where the model uses KV cache to generate the next token, and updates the KV cache with the new token.

Prefill Phase. Let $\boldsymbol{X} \in \mathbb{R}^{b \times l_{\text{prompt}} \times d}$ be the input tensor, where b is the batch size, l_{prompt} is the length of the input prompt, and d is the model hidden size. For clarity, we omit the layer index here. The key and value tensors can be computed by:

$$\boldsymbol{X_K} = \boldsymbol{X}\boldsymbol{W_K}, \boldsymbol{X_V} = \boldsymbol{X}\boldsymbol{W_V} \tag{1}$$

where $\boldsymbol{W_K}, \boldsymbol{W_V} \in \mathbb{R}^{d \times d}$ are the key and value layer weight. $\boldsymbol{X_K}, \boldsymbol{X_V}$ are cached in the memory for utilization in the subsequent decoding phase.

Decoding Phase. Let $\boldsymbol{h} \in \mathbb{R}^{b \times 1 \times d}$ be hidden state of current input token. $\boldsymbol{h_K} = \boldsymbol{h}\boldsymbol{W_K}$ and $\boldsymbol{h_V} = \boldsymbol{h}\boldsymbol{W_V}$ are the current key and value states. $\boldsymbol{h_K}$ and $\boldsymbol{h_V}$ are first employed to update the KV cache:

$$\begin{aligned} \boldsymbol{X_K} &\leftarrow \text{Concat}(\boldsymbol{X_K}, \boldsymbol{h_K}), \\ \boldsymbol{X_V} &\leftarrow \text{Concat}(\boldsymbol{X_V}, \boldsymbol{h_V}) \end{aligned} \tag{2}$$

then attention output $\boldsymbol{h_O}$ is calculated by:

$$\boldsymbol{h_O} = \text{Softmax}(\boldsymbol{h_Q}\boldsymbol{X_k^T})\boldsymbol{X_V} \tag{3}$$

where $\boldsymbol{h_Q} = \boldsymbol{h}\boldsymbol{W_Q}$ is the output of the query layer. For ease of illustration, we ignore the FFN module and other parts of the inference workflow that are not addressed in our approach.

KV Quantization The B-bit KV quantization process during the prefill phase can be expressed as follows: First, determine the minimum number z_i and the maximum number m_i in $\boldsymbol{G}_i$, where $\boldsymbol{G}_i$ is a group of number in $\boldsymbol{X}_K$ or $\boldsymbol{X}_V$. Using these numbers, compute the quantized result Q($\boldsymbol{G}_i$) for each group according to the formula:

$$Q(\boldsymbol{G}_i) = \left\lfloor \frac{\boldsymbol{G}_i - z_i}{s_i} \right\rceil, \qquad s_i = \frac{m_i - z_i}{2^B - 1} \tag{4}$$

The notation $\lfloor \cdot \rceil$ represents rounding to the nearest integer. The results from all groups are aggregated to obtain Q($\boldsymbol{X}_K$) and Q($\boldsymbol{X}_V$). During the decoding phase, the quantized Q($\boldsymbol{X_K}$) and Q($\boldsymbol{X_V}$) and the stored quantization parameters z_i and s_i are used to recover the original values. In the decoding phase, the dequantized result $\boldsymbol{X'_K}, \boldsymbol{X'_V}$ are used to calculate the attention output. $\boldsymbol{X'_K}, \boldsymbol{X'_V}$ are obtained through aggregated $\boldsymbol{G'_i}$ for each $\boldsymbol{G_i}$. $\boldsymbol{G'_i}$ can be computed using:

$$\boldsymbol{G'_i} = Q(\boldsymbol{G_i}) \cdot s_x + z_x \tag{5}$$

KV Pruning The goal of KV pruning is to find two submatrices $\boldsymbol{X}^e_K, \boldsymbol{X}^e_V \in \mathbb{R}^{b \times s \times d}$ from the full matrices $\boldsymbol{X}_K$ and $\boldsymbol{X}_V$ during the prefill phase, given a cache budget $s < n$, while maximizing performance preservation. During the decoding phase, LLMs with KV pruning only use $\boldsymbol{X}^e_K$ and $\boldsymbol{X}^e_V$ to update KV cache and generate new tokens.

$$\begin{aligned} \boldsymbol{X}^e_K &\leftarrow \text{Concat}(\boldsymbol{X^e_K}, \boldsymbol{h_K}), \\ \boldsymbol{X}^e_V &\leftarrow \text{Concat}(\boldsymbol{X^e_V}, \boldsymbol{h_V}) \end{aligned} \tag{6}$$

Quantized Pruning Quantized pruning uses KV pruning methods to obtain $\boldsymbol{X}^e_K$ and $\boldsymbol{X}^e_V$ first, and then quantizes the preserved KV states $\boldsymbol{X}^e_K$ and $\boldsymbol{X}^e_V$ to Q($\boldsymbol{X}^e_K$) and Q($\boldsymbol{X}^e_V$) using various KV quantization methods in the prefill phase. In the decoding phase, the dequantized results from Q($\boldsymbol{X}^e_K$) and Q($\boldsymbol{X}^e_V$) are used to generate new tokens.

4 Experimental Setup

Benchmarks We assess the performances of quantized pruning using LongBench (Bai et al., 2024) dataset and Needle-in-a-Haystack (Kamradt, 2023) test. We also employ RULER (Hsieh et al., 2024), a dataset with different input length and diverse types of needles across 4 task categories, to better access the impact of input length in Section 6.1. More details can be seen in Appendix A.

LLMs We use state-of-the-art open-weight LLMs, including Llama-3-8B-Instruct (Dubey et al., 2024), Mistral-7B-Instruct-v0.2 (Jiang et al., 2023). For scaling experiments in Section 6.2, we also test the performance of Llama-3.2-1B and Llama-3.2-3B (Dubey et al., 2024).

Setup We aim to comprehensively investigate the token-precision trade-off in KV cache compression. We report the ratio of compressed KV cache and the full KV cache for memory budge. For KV pruning, we employ PyramidKV (Cai et al., 2024) and SnapKV (Li et al., 2024), recognized as leading methods in diverse scenarios. We also include H2O (Zhang et al., 2024b) and StreamingLLM (Xiao et al., 2023) to evaluate the feasibility of quantized pruning in Section 5. For

Pruning Method	*LongBench*											
	Token=128				Token=512				Token=2048			
	16-bit	8-bit	4-bit	2-bit	16-bit	8-bit	4-bit	2-bit	16-bit	8-bit	4-bit	2-bit
StreamingLLM	32.1	32.2	31.7	19.1	34.6	34.5	33.9	20.7	38.1	38.2	37.8	23.8
H2O	35.6	35.6	34.7	15.8	37.5	37.4	36.7	17.7	39.8	39.7	39.0	21.1
SnapKV	35.7	35.7	35.1	16.6	40.3	40.4	39.7	20.2	41.7	41.7	41.0	22.9
PyramidKV	37.4	37.3	36.4	17.5	40.3	40.3	39.6	20.9	41.8	41.8	41.3	23.6
Pruning Method	***Needle-in-a-Haystack***											
	Token=128				Token=512				Token=2048			
	16-bit	8-bit	4-bit	2-bit	16-bit	8-bit	4-bit	2-bit	16-bit	8-bit	4-bit	2-bit
StreamingLLM	27.7	27.7	27.5	30.9	35.3	35.3	35.5	37.3	66.4	66.5	66.4	61.8
H2O	46.9	46.6	46.8	36.4	91.2	91.1	91.0	54.8	100	100	100	74.1
SnapKV	83.7	83.7	82.5	55.9	97.4	97.4	97.2	66.3	100	100	100	78.1
PyramidKV	98.9	98.9	98.8	67.5	100	100	100	78.6	100	100	100	79.6

Table 1: Feasibility of quantized pruned tokens on LongBench and Needle-in-a-Haystack with Llama-3-8B-Instruct as backbone model. We use four KV pruning methods to retain 128, 512 and 2048 tokens, and report the results of further quantization.

KV quantization, we adopt KIVI (Liu et al., 2024c) as the default method due to its stability and broad compatibility. Moreover, in Section 6.3, we examine the effects of quantization strategies from FlexGen (Sheng et al., 2023) and KVQuant (Hooper et al., 2024) for a comprehensive comparison. We use HQQ (Badri and Shaji, 2023) framework to perform quantization on KV cache. More details can be seen in Appendix B.

5 Optimal Token-Precision Trade-Off

In this section, we aim to find the optimal token-precision trade-off in KV cache compression. We frist examine the feasibility of combining KV pruning and quantization(Q1). Subsequently, we explore the best optimal allocation strategy between precision and token under varying memory budgets(Q2).

> Q1. Is it feasible to quantize pruned KV cache for a lower compression rate?

We first evaluate the feasibility of quantized pruned KV cache as a prerequisite for exploring the token-precision trade-off. We use Llama-3-8B-Instruct and evaluate various KV pruning methods on the LongBench and NIAH. We report the results of quantizing the remaining tokens to different precision levels after applying KV pruning.

From Table 1, we observe that *it is feasible to quantize pruned KV cache for a lowe compression rate.* For most KV pruning methods we evaluate, further quantizing the preserved tokens to as low as 4-bit precision results in minimal performance degradation, but quantizing to 8-bit precision shows negligible impact. However, reducing precision to 2-bit leads to a drastic performance decline across most KV pruning methods. This observation holds consistently across different KV pruning methods and varying numbers of preserved tokens.

Compared with precision, reducing the number of preserved tokens leads to more significant performance degradation. Specifically, when the number of preserved tokens is reduced to 1/4 (from 2048 to 512), all KV pruning methods experience a noticeable performance drop. In contrast, when the precision is reduced to 1/4 (from 16-bit to 4-bit), which has the same memory budget as token dimension, the performance degradation is relatively mild. This suggests that, under the same memory budget, tokens might have a more significant impact on the results compared to precision.

> Q2. What is the optimal allocation strategy between precision and token under varying memory budgets?

Observing that KV pruning and KV quantization can be effectively combined, we further investigate that, given a fixed memory budget, how to balance the trade-off between number of preserved tokens and precision to achieve optimal performance. To

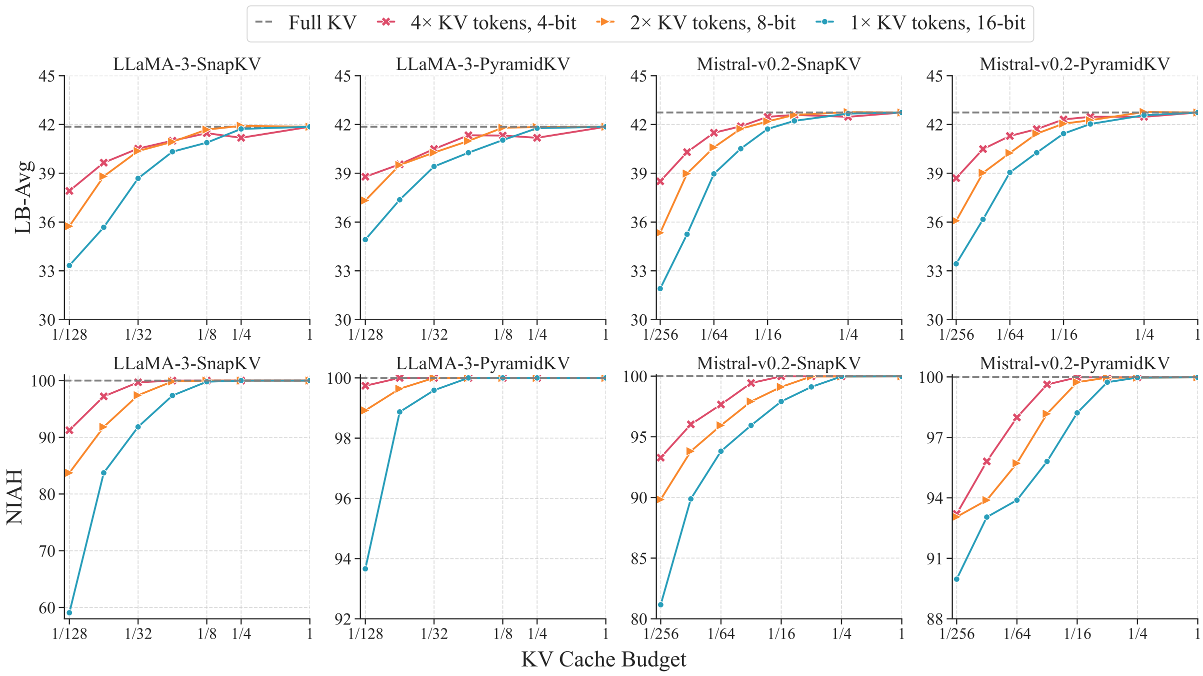

Figure 2: The token-precision trade-off under varying memory budgets on LongBench and NIAH. We report the results of SnapKV-based and PyramidKV-based quantized pruning on Llama-3 and Mistral-v0.2.

this end, we evaluate the performance of quantized pruning using two leading KV pruning methods, SnapKV and PyramidKV, across different memory budgets on LongBench and NIAH.

Specifically, we compare three configurations with approximately equivalent memory usage: 1) Using standalone KV pruning to retain $1\times$ tokens in 16-bit precision. 2) Quantized pruning by retaining $2\times$ tokens in 8-bit precision. 3) Quantized pruning by retaining $4\times$ tokens in 4-bit precision.

As shown in Figure 2, we observe that *quantized pruning, which preserves more tokens at a lower precision, consistently outperforms standalone KV pruning methods across various budgets.* For the NIAH task, the improvements from quantized pruning are particularly pronounced. This may be attributed to that quantized pruning can cover more tokens for retrieval under the same memory budget compared to standalone KV pruning.

In high-budget scenarios, the 8-bit strategy tends to deliver slightly better performance, which may due to the number of tokens at this budget is already quite large. In low-budget scenarios, such as 1/128 KV cache budget, storing more tokens at 4-bit precision yields superior results, highlighting the importance of token coverage when resources are constrained. Overall, using lower precision to preserve more tokens under a limited budget results in notable performance gains, compared to standalone KV pruning methods that use full precision to store fewer tokens.

Summary We demonstrate that storing ***more tokens*** in the KV cache with ***lower precision*** can significantly enhance the long-context performance of LLMs under fixed KV cache memory budget.

6 Further Analysis

In this section, we further investigate series of key aspects regarding token-precision trade-off, including the impact of quantized pruning on various downstream task types and input lengths, model scaling effect, ablation on quantization strategies and fine-grained exploration of layer-wise quantized pruning.

6.1 Impact on Task Types and Input Lengths

Task Types To further investigate the token-precision trade-off in different task types, we evaluate PyramidKV-based quantized pruning on six task types from LongBench and the 8K subset of the RULER dataset. We use PyramidKV with 512 retained tokens as the baseline, and explore the token-precision trade-off under this fixed memory budget, as this setting exhibits minimal performance differences across three precision levels, making it easier to assess the impact of task types.

As illustrated in Table 2, we observe that the performance of quantized pruning is remarkably

Models	Token	Bit	Task Types						
			SQA	MQA	SUMM	Fewshot	Syn.	Code	RULER-8k
Llama-3-8B-Instruct	512	16	28.2	31.9	23.5	67.6	37.7	57.6	67.5
	1024	8	29.6	33.1	24.3	67.9	37.4	58	74.9
	2048	4	30.7	32.5	25.3	68.8	37.2	57.6	82.2
Mistral-7B-Instruct-v0.2	512	16	33.7	27.3	24.3	65.6	41.75	54	53.1
	1024	8	34.2	29	25.6	66.4	43.73	54.8	62.1
	2048	4	35.2	28.14	26.6	66.9	43.08	55.4	73.6

Table 2: The token-precision trade-off in different task types. We report the results of 6 task types in LongBench and 8k subset of RULER. We use PyramidKV-based quantized pruning.

consistent across different task types. Specifically, lower precision, which retains more tokens in KV cache, leads to substantial performance improvements in the RULER task, which heavily relies on retrieving content from the input. Tasks with high retrieval demands, such as Summarization and Single-Doc QA, also show noticeable gains with quantized pruning, particularly when $4\times$ tokens are preserved at 4-bit precision.

For tasks requiring more reasoning rather than intensive retrieval, such as Code Completion, Synthetic and Multi-Doc QA, the benefits of trading precision for more tokens are less pronounced. In these cases, storing fewer tokens with higher precision generally performs better. For example, using 1024 tokens in 8-bit precision achieves the hightest score of 58 in Code task with Llama-3.

Input Lengths To evaluate the token-precision trade-off across various input lengths, we conduct experiments on subsets with different input length of the RULER dataset. Additionally, we analyze LongBench by grouping its data based on input length. The results is shown in Figure 3 and more detailed information can be found in Appendix A.

Our observations are as follows: *quantized pruning consistently outperforms standalone KV eviction methods across various input length, regardless of the models and task types.* Within the same dataset, scores decrease as input length increases; however, the relative differences among different compression methods remain similar across varying input lengths. Moreover, quantized pruning achieves significant performance improvements across all input lengths for retrieval demanded tasks like RULER.

6.2 Scaling Effect on Quantized Pruning

To investigate the impact of model scaling on quantized pruning, we conducted experiments on

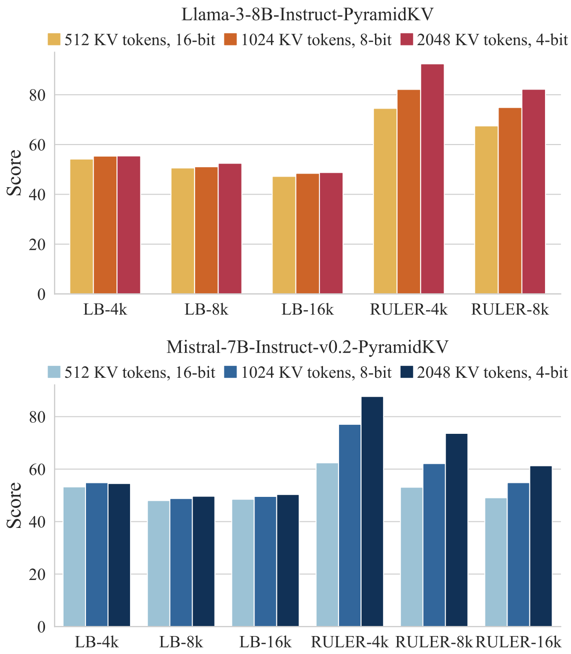

Figure 3: The token-precision trade-off in different input lengths. We report the results of LongBench and three subsets of RULER. We use PyramidKV-based quantized pruning.

three models from the Llama series: Llama3-8B, Llama3.2-3B, and Llama3.2-1B. For both the Base models and Instruct models, we evaluated their performance on LongBench under two fixed KV cache budgets: 1/16 and 1/64.

As shown in Figure 4, we observe that quantized pruning consistently achieves better performance across all scaling levels. The performance gap between quantized pruning and standalone KV pruning methods remains relatively stable across different model scales. Notably, when the KV cache budget is relative small to 1/64 , the performance improvement brought by quantized pruning is higher compared to 1/16 KV cache budget, which aligns with the conclusions we observed earlier in Q2. For

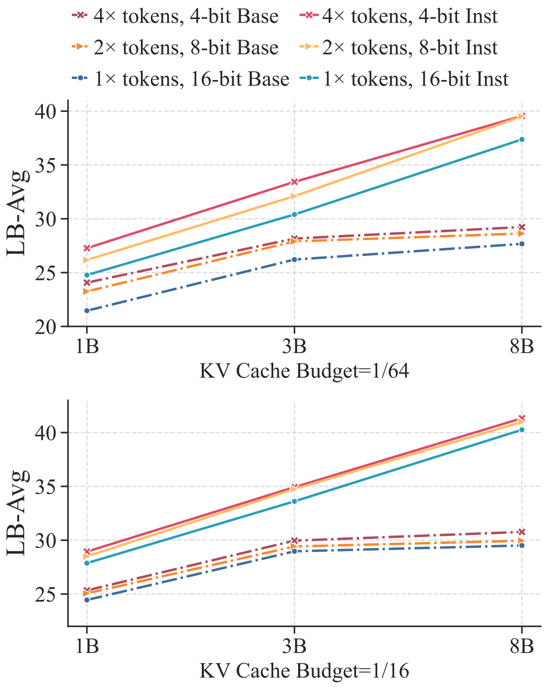

Figure 4: Scaling effect on Llama family models, with PyramidKV-based quantized pruning. All models are under fixed ratio of KV cache budget.

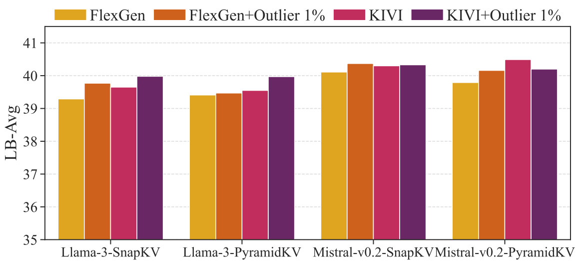

Figure 5: Ablation of quantization strategies on quantized pruning, remaining 512 KV tokens in 4-bit.

Base models, although the performance improvement from scaling is smaller compared to Instruct models, quantized pruning still provides a noticeable performance boost.

These findings highlighting the robustness and effectiveness of quantized pruning across model scaling.

6.3 Ablation on Quantization Strategies

While there has been extensive research on strategies for KV cache quantization, it remains unclear whether existing quantization strategies remain effective when combined with KV pruning methods. In this section, we aim to investigate the impact of KV quantization strategies and group size on quantized pruning, and present our results in Figure 5 and Table 3.

Model	Method	GroupSize		
		32	64	128
Llama-3	SnapKV	40.42	39.55	38.87
	PyramidKV	40.33	39.65	38.91
Mistral-v0.2	SnapKV	40.45	40.30	40.09
	PyramidKV	40.31	40.49	40.03

Table 3: The impact of group size for quantized pruning on LongBench, remaining 512 KV tokens in 4-bit.

Quantization methods We explore the methods in FlexGen (Sheng et al., 2023), KIVI (Liu et al., 2024c), and KVQuant (Hooper et al., 2024). To elaborate, for the FlexGen methods, KV quantization is applied to both the key and value caches along the token dimension, grouping every 64 elements without filtering outlier numbers. We modify the FlexGen by (1) filtering 1% of outlier numbers in both the key and value caches, as mentioned in KVQuant (2) quantizing the key along the channel dimension, as in KIVI and (3) combining (1) and (2). These correspond to the results labeled as FlexGen+Outlier 1%, KIVI, and KIVI+Outlier 1% in the Figure 5.

We can observe that none of the quantization strategies show significant performance degradation when combined with KV pruning methods, demonstrating the relative stability of quantized pruning. The KIVI method consistently outperforms FlexGen across various models and KV pruning methods. The improvement is particularly pronounced for PyramidKV on the Mistral model, underscoring the significance of quantizing key states along the channel dimension. Filtering 1% of outlier numbers proves effective for the FlexGen strategy but yields limited improvements for KIVI. It shows some benefit on Llama models but can result in negative gains on the Mistral model.

Overall, KIVI demonstrates strong performance when combined with KV pruning methods, while other KV quantization strategies also maintain good results, highlighting the stability of quantized pruning.

Group Size We then analyzed the impact of group size during KV quantization. Using the SnapKV and PyramidKV methods to retain 512 tokens, we experimented with 4-bit quantization and observed the performance variations when the group sizes were set to 32, 64 and 128.

As shown in Table 3, smaller group sizes lead to performance improvements at the cost of higher memory usage. Reducing the group size from 128 to 64 resulted in a notable improvement, but further decreasing it from 64 to 32 yielded minimal gains for the Mistral model. Therefore, we set the default quantization group size to 64 to balance performance and memory usage in our experiments.

6.4 Exploration on Layer-Wise Quantized Pruning

Inspired by the observation that different layers may have varying requirements for the number of tokens in PyramidKV (Cai et al., 2024) and PyramidInfer (Yang et al., 2024a), we further investigate whether the demands for precision and preserved tokens are consistent across layers. To explore this, we use the best-performing configuration from previous experiments, 4-bit precision with $4\times$ tokens, as the baseline and compare it against layer-wise configurations adopting 8-bit precision with $2\times$ tokens and 16-bit precision with $1\times$ tokens. Using SnapKV as the KV pruning method, we present the results for Llama-3 and Mistral-v0.2 under two budget constraints in the Figure 6. Configurations are modified every 4 layers for the initial and final layers, while intermediate layers are reconfigured every 8 layers. The x-axis indicates the layers where the modified configurations are applied, while the y-axis shows the relative change to the baseline (4-bit precision with $4\times$ tokens) on LongBench and RULER-4k.

Initially, it is evident that for most layers, transitioning from $4\times$ tokens with 4-bit precision to higher precision and fewer tokens results in a performance decline under constrained KV cache budgets. Specifically, the shift to 8-bit shows a relatively minor performance drop, whereas moving to 16-bit with fewer preserved tokens leads to a more significant decrease. These layers-wise trade-off conclusions are consistent with our experiments before.

Notably, modifying intermediate layers causes a drastic performance decline, while adjustments made at the initial and final layers result in comparatively smaller performance reductions. This effect is especially pronounced in retrieval-related tasks such as RULER-4k, where significant performance differences are observed. On LongBench, changes are less significant, with a notable performance drop only observed at 16-bit precision. These findings highlight that, under the same memory budget, *preserving more tokens in intermediate layers is crucial for the performance*, while the token-precision trade-off in the initial and final layers exerts a more balanced influence on the results.

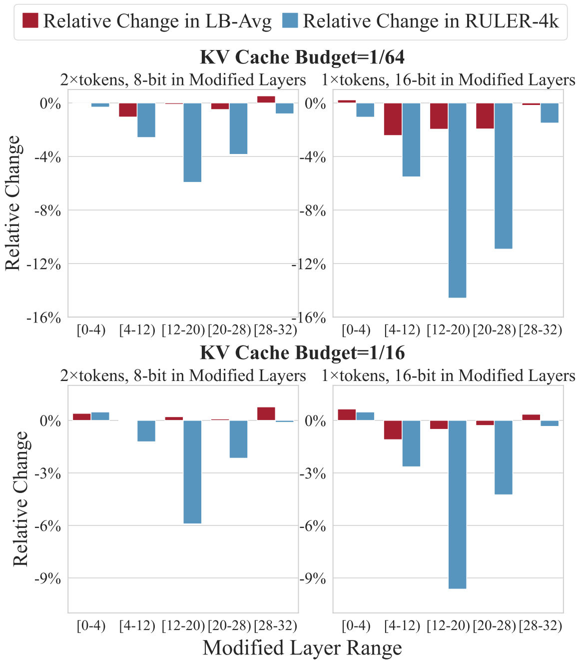

Figure 6: The results of layer-wise quantized pruning on Llama-3-8B-Instruct, with SnapKV as pruning method. We use $4\times$ KV token 4-bit as baseline and report the relative change.

7 Conclusion

We investigate a series of critical yet unexplored questions regarding the effectiveness and feasibility of token-precision trade-off in KV cache compression. Through comprehensive experiments, we demonstrate that storing more tokens in the KV cache with lower precisioncan significantly enhance the long-context performance of LLMs, and demonstrating robust performance across diverse input lengths, downstream tasks, with particularly significant gains in retrieval tasks. Moreover, we find quantized pruning demonstrates strong feasibility across different KV pruning methods, quantization strategies, and model scales. Our analysis sheds light on the token-precision trade-off of KV cache memory optimization, offering valuable insights into designing more efficient compression strategies. We hope this work deepens our understanding of KV cache compression and inspires future research.

Limitations

While our work demonstrates the effectiveness of KV compression through trade-offs between token and precision dimensions, other potential dimensions, such as head and layer, remain unexplored. Investigating the feasibility of combining these dimensions with token and precision for a more substantial compression potential represents an avenue for future research. Additionally, the current implementation of quantized pruning suffers from inefficiencies in dequantizing the KV cache, hindering the full realization of speedup benefits from the memory savings. In future work, we aim to address this issue by optimizing the implementation, such as integrating fusion operators to combine dequantization with matrix multiplication.

References

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

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Hicham Badri and Appu Shaji. 2023. Half-quadratic quantization of large machine learning models.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. 2024. Reducing transformer key-value cache size with cross-layer attention. *Preprint*, arXiv:2405.12981.

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

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A simple and effective l_2 norm-based strategy for kv cache compression. *Preprint*, arXiv:2406.11430.

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. 2024. Qaq: Quality adaptive quantization for llm kv cache. *Preprint*, arXiv:2403.04643.

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

Razvan-Gabriel Dumitru, Paul-Ioan Clotan, Vikas Yadav, Darius Peteleaza, and Mihai Surdeanu. 2024. Change is the only constant: Dynamic llm slicing based on layer redundancy. *arXiv preprint arXiv:2411.03513*.

Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev. 2019. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. *arXiv preprint arXiv:1906.01749*.

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

Yao Fu. 2024. Challenges in deploying long-context transformers: A theoretical peak performance analysis. *Preprint*, arXiv:2405.08944.

Yu Fu, Zefan Cai, Abedelkadir Asi, Wayne Xiong, Yue Dong, and Wen Xiao. 2024. Not all heads matter: A head-level kv cache compression method with integrated retrieval and reasoning. *arXiv preprint arXiv:2410.19258*.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: A long-range pretrained language model for code completion. In *International Conference on Machine Learning*, pages 12098–12107. PMLR.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Preprint*, arXiv:2401.18079.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

Greg Kamradt. 2023. Needle in a haystack - pressure testing llms. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *Preprint*, arXiv:2403.05527.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

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. *Preprint*, arXiv:2404.14469.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. 2024b. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024c. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32332–32344. PMLR.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state rnns. *Preprint*, arXiv:2401.06104.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*, volume 5, pages 606–624. Curan.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Siyu Ren and Kenny Q. Zhu. 2024. On the efficacy of eviction policy for key-value constrained generative language model inference. *Preprint*, arXiv:2402.06262.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

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

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. *Preprint*, arXiv:2303.06865.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You only cache once: Decoder-decoder architectures for language models. *ArXiv*, abs/2405.05254.

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

Daniel Waddington, Juan Colmenares, Jilong Kuang, and Fengguang Song. 2013. Kv-cache: A scalable high-performance web-object cache for manycore. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 123–130.

Haoyi Wu and Kewei Tu. 2024. Layer-condensed kv cache for efficient inference of large language models. *Preprint*, arXiv:2405.10637.

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

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

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

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. 2024b. No token left behind: Reliable kv cache compression via importance-aware mixed precision quantization. *Preprint*, arXiv:2402.18096.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024a. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *Preprint*, arXiv:2405.03917.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2024b. H2o: heavy-hitter oracle for efficient generative inference of large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, et al. 2021. Qmsum: A new benchmark for query-based multi-domain meeting summarization. *arXiv preprint arXiv:2104.05938*.

Anni Zou, Wenhao Yu, Hongming Zhang, Kaixin Ma, Deng Cai, Zhuosheng Zhang, Hai Zhao, and Dong Yu. 2024. Docbench: A benchmark for evaluating llm-based document reading systems. *Preprint*, arXiv:2407.10701.

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

A Datasets

LongBench LongBench (Bai et al., 2024) includes 17 datasets covering 6 categories of tasks, which can be divided into single-document QA (Dasigi et al., 2021; Kočiskỳ et al., 2018), multi-document QA (Yang et al., 2018; Ho et al., 2020), summarization (Huang et al., 2021; Fabbri et al., 2019; Zhong et al., 2021), few-shot learning (Gliwa et al., 2019; Joshi et al., 2017; Li and Roth, 2002), synthetic, and code generation (Guo et al., 2023; Liu et al., 2023). LongBench features an average input length ranging from 1,235 to 18,409 tokens. For inputs exceeding the model's context window length(8k for Llama-3-8B-Instruct (Dubey et al., 2024), we split the data and only take the beginning and end segments of the input to fill the context window length. Additionally, we reserve sufficient space for newly generated tokens based on the specific type of subdataset. For Q4, we select datasets with sufficient data to cover three input length ranges: (<4k, 4k 8k, and >8k). These datasets include MultiFieldQA-en, 2WikiMultihopQA, GovReport, TREC, TriviaQA, SAMSum, and RepoBench-P, representing a variety of task types. We refer to the three subsets as LB-4k, LB-8k, and LB-16k, respectively.

NIAH Needle-in-a-Haystack(NIAH) (Kamradt, 2023) is a challenging pressure test designed to assess the ability of models to accurate identify and retrieve relevant information from lengthy context. NIAH randomly inserts key information into an arbitrary position within a long essay. In our setup, we use PaulGrahamEssays as the haystack and the sentence "The best thing to do in San Francisco is eat a sandwich and sit in Dolores Park on a sunny day." as the needle, which is the default setting of NIAH. We vary the essay length from 1,000 tokens up to the models' context window limits, increasing by 100 tokens per step for Llama-series models and 400 tokens per step for Mistral. The results are reported as the average score across all tests.

RULER RULER (Hsieh et al., 2024) generates synthetic examples to evaluate long-context language models with configurable sequence lengths and varying task complexities. It includes four task categories: Retrieval, Multi-hop Tracing, Aggregation, and Question Answering. The dataset comprises six subsets with input lengths of 4K, 8K, 16K, 32K, 64K and 128K tokens. In our experiments, we use the 4K, 8K and 16K subsets to test the models within their context window limits.

B Experiment Setup

Memory Budgets We report the ratio of compressed KV cache and the full KV cache for memory budge. The full KV cache for Llama-3 is 8k KV tokens in 16-bit on both LongBench and NIAH, while for Mistral-v0.2 is 16k KV tokens on LongBench and 32k KV tokens on NIAH in 16-bit.

KV eviction methods We retain the last 32 tokens for StreamingLLM (Xiao et al., 2023), H2O (Zhang et al., 2024b), and SnapKV (Li et al., 2024), while keeping 8 tokens for PyramidKV (Cai et al., 2024), as recommended in the corresponding paper (Cai et al., 2024). For other settings, we adopt the default configurations from the PyramidKV codebase.

KV quantization We utilize HQQQuantizedCache from Huggingface and adjust the group dimensions of keys and values to implement grouped quantization strategies from FlexGen (Sheng et al., 2023) and KIVI (Liu et al., 2024c). We use 64 as the default group size which is suggested in FlexGen (Sheng et al., 2023).In the experiments involving outlier filtering, we exclude numbers in the KV cache with a absolute value exceeding 6 from quantization, which roughly corresponds to the top 1% of outliers based on our validation set analysis.

C More results in Experiments

Layer-Wise Quantized Pruning We alse report the results for Mistral-v0.2 in Figure 7, we can see the layer-wise results are similiar to Llama-3.

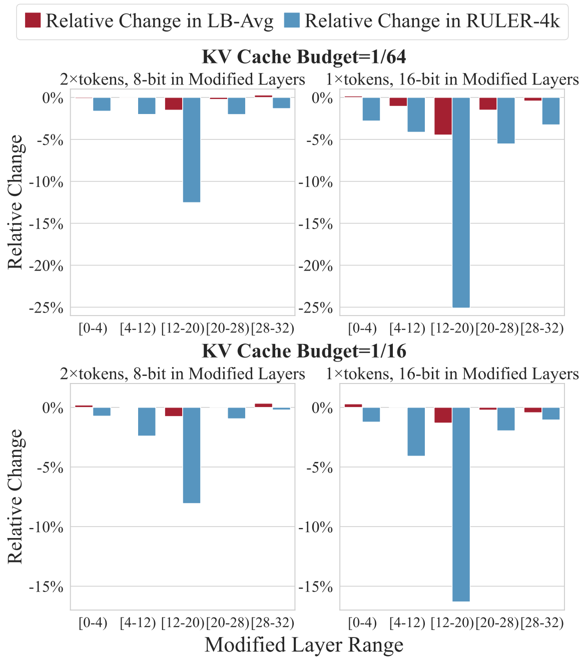

Figure 7: The results of Layer-Wise Quantized Pruning on Mistral-7B-v0.2-Instruct, with SnapKV as pruning method. We use $4\times$ KV token 4-bit as baseline and report the relative change.