

SQuat: Subspace-orthogonal KV Cache Quantization

Hao Wang*, Ligong Han*, Kai Xu, Akash Srivastava
Red Hat AI Innovation

Abstract

The key-value (KV) cache accelerates LLMs decoding by storing KV tensors from previously generated tokens. It reduces redundant computation at the cost of increased memory usage. To mitigate this overhead, existing approaches compress KV tensors into lower-bit representations; however, quantization errors can accumulate as more tokens are generated, potentially resulting in undesired outputs. In this paper, we introduce **SQuat** (Subspace-orthogonal KV cache **q**uantization). It first constructs a subspace spanned by query tensors to capture the most critical task-related information. During key tensor quantization, it enforces that the difference between the (de)quantized and original keys remains orthogonal to this subspace, minimizing the impact of quantization errors on the attention mechanism’s outputs. SQuat requires no model fine-tuning, no additional calibration dataset for offline learning, and is grounded in a theoretical framework we develop. Through numerical experiments, we show that our method reduces peak memory by $2.17\times \sim 2.82\times$, improves throughput by $2.45\times \sim 3.60\times$, and achieves more favorable benchmark scores than existing KV cache quantization algorithms. Our code is available at: <https://github.com/Red-Hat-AI-Innovation-Team/SQuat>

1 Introduction

The future of AI technology should be accessible to everyone and operate efficiently. Just as computers evolved from bulky, expensive machines into widely available, affordable tools, AI is undergoing a similar transformation. Today, large language models (LLMs) are increasingly used in many applications, yet challenges remain. Unlike traditional software applications that execute tasks almost instantly, LLMs—even during inference—require significant computational resources to process user queries. To reduce compute costs in auto-regressive decoding, causal language models often use the key-value (KV) cache. This cache stores previously computed keys and values for the attention mechanism, allowing the model to avoid redundant calculations. In essence, it is just like how humans remember the context of a conversation—we do not start from scratch with every reply, but build on what was said before.

A major bottleneck introduced by the KV cache is the increased GPU memory consumption and the latency from frequent data transfers between memory and compute units (Lienhart, 2024; Verma & Vaidya, 2023). The challenge is amplified in modern reasoning models (OpenAI, 2024; Guo et al., 2025), which often require long responses to “think” through problems before arriving at a final answer. Moreover, inference-time scaling methods further exacerbate these issues by increasing context lengths or the number of concurrent decoding paths (Lightman et al., 2023; Rush & Ritter, 2024). To address this challenge, a burgeoning line of research investigates strategies to optimize KV cache management, mitigate memory constraints, and accelerate inference speed (see e.g., Kwon et al., 2023; Dubey et al., 2024; Liu et al., 2024a). Within this effort, KV cache quantization is a crucial approach that compresses stored keys and values into lower-precision formats. It eliminates the need for re-training or fine-tuning LLMs, preserves full-context information, and remains compatible with methods like token pruning and model compression.

*Equal contribution. Correspondence to: {hao-wang, lihan}@redhat.com

Existing approaches to KV cache quantization typically treat it as a lossy data compression problem (see e.g., Yang et al., 2024; Liu et al., 2024e; Kang et al., 2024; Hooper et al., 2024b). They aim to minimize the discrepancy between full-precision KV tensors (e.g., FP16) and their low-bit representations (e.g., INT2) without explicitly accounting for their effect on the model’s next-token prediction. Consequently, while they preserve the numerical fidelity of KV tensors to some extent, task-critical information—necessary for accurately responding to user queries—may be lost. Moreover, quantization errors from early tokens can accumulate and amplify during decoding, causing later tokens to deviate from expected outputs and eventually leading to degraded response quality or misalignment with user queries. This motivates a fundamental question:

How can we preserve task-relevant information during KV cache quantization to ensure that the LLM generates outputs as if using full-precision KV tensors?

In this paper, we introduce **SQuat** (Subspace-orthogonal KV cache **quantization**), a new approach for KV cache quantization. It is motivated by our two observations:

- *Observation 1.* The attention scores in transformer architectures are calculated by taking the inner product between the query tensors of the new token and the key tensors of all past tokens. Thus, when quantizing key tensors, the primary objective should be to preserve their inner products with future tokens’ query tensors, rather than merely minimizing the difference between the original and (de)quantized key tensors.
- *Observation 2.* Query tensors tend to lie within a subspace whose dimension is significantly smaller than the hidden dimension. Hence, even without knowing future tokens’ query tensors at the time of quantizing the current token’s key tensors, we can still leverage this subspace to guide the quantization process.

Building on these insights, *SQuat* first constructs a task-relevant subspace using the query tensors from all tokens in user-provided prompts. It then quantizes each token’s key tensors, ensuring that the residuals (i.e., differences between the (de)quantized and original key tensors) stays as orthogonal to this subspace as possible, reducing the quantization error’s effect on critical task information (see Figure 1 for our pipeline). *SQuat* requires no model training, no additional calibration data for offline learning, and is grounded in a theoretical framework we develop.

SQuat is based on an optimization that balances the trade-off between maintaining quantization fidelity and keeping residuals orthogonal to the query subspace. We introduce an iterative algorithm that approximately solves this optimization problem. At each step, an element (or block of elements) of the key tensors is quantized, followed by an update to the remaining elements. This update minimizes the quantization error’s impact on inner products with vectors in the query subspace. We prove that the optimal update rule has a closed-form expression and present an efficient algorithm for computing it. This effort significantly reduces computational complexity and enables *SQuat* to run on-the-fly.

We conduct extensive numerical experiments to evaluate *SQuat* and compare it against tuning-free baselines. Specifically, we apply these methods to quantize the KV cache of four LLMs: Llama-2-7B (Touvron et al., 2023), Llama-3.1-8B-Instruct (Dubey et al., 2024), Mistral-7B-Instruct-v0.3 (Jiang et al., 2023), and DeepSeek-R1-Distill-Llama-8B (Guo et al., 2025). We evaluate their performance across two benchmark families: (1) a set of challenging reasoning tasks that often need long responses before reaching the final answer, and (2) LongBench tasks that focus on long-context understanding, such as document QA and summarization, which require long input windows (Bai et al., 2023). Together, these benchmarks cover 14 tasks. *SQuat* achieve more favorable performance than existing tuning-free baselines. When applied to the Llama-2-7B model, *SQuat* reduces peak memory usage by $2.17\times \sim 2.82\times$ compared to the default FP16 format, resulting in a throughput improvement of $2.45\times \sim 3.60\times$.

1.1 Related Work

KV cache management. Recent research has explored advanced KV cache management techniques to optimize inference efficiency and reduce memory usage in LLMs. One strategy focuses on token pruning, eviction, or merging (Xiao et al., 2023b; Beltagy et al., 2020; Kim et al., 2022; Ge et al., 2023; Liu et al., 2024d; Han et al., 2023; Zhang et al., 2023; Lee et al., 2025; Wang et al., 2024; Wan et al., 2024; Zhang et al., 2024b; Jiang et al., 2024), where older or less relevant tokens are removed from the KV cache, or similar and redundant tokens are merged. Some methods (Cai et al., 2024; Li et al., 2024) allow for selective eviction of KV tensors across different attention layers rather than uniformly pruning KV tensors for the same tokens at all layers. A related technique (Xu et al., 2024) prunes specific key tensor channels instead of entire tokens, leveraging query information to guide the pruning process. Beyond pruning-based methods, there are other techniques to reduce the memory overhead of storing the KV cache. For example, Tang et al. (2024); Hooper et al. (2024a); Ribar et al. (2023) propose storing the full KV cache but dynamically load only the relevant keys and values during inference. These approaches, along with advancements in system and architecture design, are orthogonal to our method on KV cache quantization and can potentially be combined together to enhance GPU utilization.

KV cache compression. There has been active research on KV cache quantization (Tao et al., 2024; Yang et al., 2024; Byun et al., 2024; Zhang et al., 2024a; Liu et al., 2024e; Kang et al., 2024; Hooper et al., 2024b; Sheng et al., 2023; Zandieh et al., 2024; He et al., 2024; Yue et al., 2024; Liu et al., 2024c; Chang et al., 2024; Zhang & Shen, 2024; Duanmu et al., 2024; Zhao et al., 2024). For example, Xiao et al. (2023a) Sheng et al. (2023) introduce 8-bit and 4-bit group-wise quantization for both model weights and KV cache. More recently, Liu et al. (2024e); Kang et al. (2024); Hooper et al. (2024b) explore per-channel quantization for key tensors and per-token quantization for value tensors, as key tensors show significant variation across channels, whereas values do not. Similarly, Duanmu et al. (2024) propose rearranging KV cache channels to enhance similarity within quantization groups. Hooper et al. (2024b) further introduce pre-RoPE quantization, outlier isolation, and non-uniform quantization. Kim et al. (2024) develop an input-independent dictionary and represent KV tensors as sparse linear combinations of its elements. These advancements enable 2-bit quantization while maintaining performance comparable to non-quantized baselines. Among these approaches, Liu et al. (2024e); Kang et al. (2024); He et al. (2024), like our method, do not require fine-tuning or a calibration dataset. Liu et al. (2024e) present a hardware-friendly GPU implementation, inspiring a new quantization method integrated into Hugging Face Transformers (Turganbay, 2024). Kang et al. (2024) reduce quantization error by storing a low-rank matrix and a sparse matrix. He et al. (2024) consider first identifying salient tokens and then applying mixed-precision quantization to compress the KV cache. As we will discuss in Section 3.1, these compression-based approaches do not explicitly account for quantization errors’ impact on attention outputs. In contrast, our method preserves the inner product between key tensors and future tokens’ query tensors, minimizing quantization-induced deviations in LLM outputs. A comprehensive comparison with these methods will be provided in our numerical experiments (Section 4).

Efficient system and architecture designs. Optimizing computer systems and transformer architectures for efficient LLM inference has been an active research area. One line of research focuses on system design for optimizing KV cache management on GPUs (Jin et al., 2023; Yu et al., 2022; NVIDIA, 2023). Among them, Kwon et al. (2023) propose paged attention, a block-based dynamic memory allocation system that serves as the foundation of vLLM, a widely adopted library for LLM inference and serving. Another research direction explores modifications to attention mechanisms in transformer-based LLMs to reduce KV cache size (Child et al., 2019; Katharopoulos et al., 2020; Choromanski et al., 2020; Wu & Tu, 2024; Liu et al., 2024b;a; Yuan et al., 2025; Yu et al., 2024). For example, multi-query and grouped-query attention mechanisms maintain multiple query heads while sharing a single set of KV pairs across all heads or within groups of heads (Ainslie et al., 2023; Shazeer, 2019). This architecture has been incorporated into many open-source LLMs, including Llama (Dubey et al., 2024) and PaLM (Chowdhery et al., 2023).

Model compression. The study of pruning, quantization, and sparsification in neural networks has a long history (Gupta et al., 2015; Frankle & Carbin, 2018; Frantar & Alistarh, 2022), originating from early efforts (LeCun et al., 1989; Hassibi et al., 1993) to optimize computational efficiency and memory usage while preserving accuracy. Recently, these techniques have gained increasing attention in the context of LLMs, which have grown to hundreds of billions of parameters, demanding substantial compute resources for both training and inference (Frantar & Alistarh, 2023; Sun et al., 2023; Frantar et al., 2022; Xiao et al., 2023a; Yao et al., 2022; Dettmers et al., 2022; Kurtić et al., 2023; Malinovskii et al., 2024; Ashkboos et al., 2024). Our work focuses on quantizing the KV cache rather than model weights. Unlike static weight matrices, which can be pre-quantized and stored before inference, KV tensors are generated dynamically as new tokens arrive. This streaming nature demands quantization to be performed on-the-fly, requiring methods that minimize computational overhead and latency to ensure efficient decoding.

2 Background

(Masked) Attention mechanism. For the sake of illustration, we assume batch size is 1 and the attention mechanism has a single head. A transformer consists of L layers and each layer contains a multi-head attention and a feed-forward network. Suppose the input token embeddings are: $\mathbf{X} \in \mathbb{R}^{n \times d}$.

$$\text{Query matrix: } \mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \text{Key matrix: } \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \text{Value matrix: } \mathbf{V} = \mathbf{X}\mathbf{W}^V,$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d \times d}$ are the projection matrices. The attention mechanism outputs: $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} + \mathbf{M}\right) \mathbf{V}$, where the mask $\mathbf{M}$ is a strictly upper triangular matrix, with zeros on and below the diagonal and $-\infty$ in every element above the diagonal.

Inference-time workflow. In the *prefill phase*, let $\mathbf{X}_0 \in \mathbb{R}^{l_{\text{prompt}} \times d}$ represent the input tensor, where l_{prompt} is the length of the user’s input prompt and d denotes the model’s hidden size. During this phase, the key ($\mathbf{K}$) and value ($\mathbf{V}$) tensors are computed and cached. This is followed by the *decoding phase*, in which the LLM predicts the next token sequentially, conditioned on the tokens generated so far. Note that LLMs only compute the query, key, and value for the newly generated token. The new key and value are then concatenated with the previously cached tensors. This caching mechanism significantly reduces computational overhead, as it avoids recomputing the key and value tensors for all prior tokens at every decoding step, resulting in faster inference. However, the KV cache may become a memory bottleneck when handling long contexts or large batch sizes (see Appendix B). Compressing the KV cache—through methods such as quantization—offers a natural solution to this challenge.

Quantization. In practice, quantization is often applied to a group of numbers $x_1, \dots, x_n$ simultaneously. In the context of KV cache quantization, these numbers can be elements of the KV cache for the same token across hidden dimensions (per-token quantization) or elements from multiple tokens within the same hidden dimension (per-channel quantization). Quantization begins by determining $m = \min\{x_i\}_{i=1}^n$ and $M = \max\{x_i\}_{i=1}^n$, and then maps each number to a b -bit integer using:

$$\text{qtz}(x_i) = \left\lfloor \frac{x_i - m}{\Delta} \right\rfloor \quad \text{with } \Delta = \frac{M - m}{2^b - 1}.$$

Here m and Δ are referred to as the zero-point and scaling factor, respectively. The operator $\lfloor \cdot \rfloor$ rounds to the nearest integer. After quantization, the b -bit integer falls within the range $[0, 2^b - 1]$. The quantized values, along with the zero-point and scaling factor, are stored to enable dequantization: $\text{deq}(\bar{x}_i) = \bar{x}_i \times \Delta + m$.

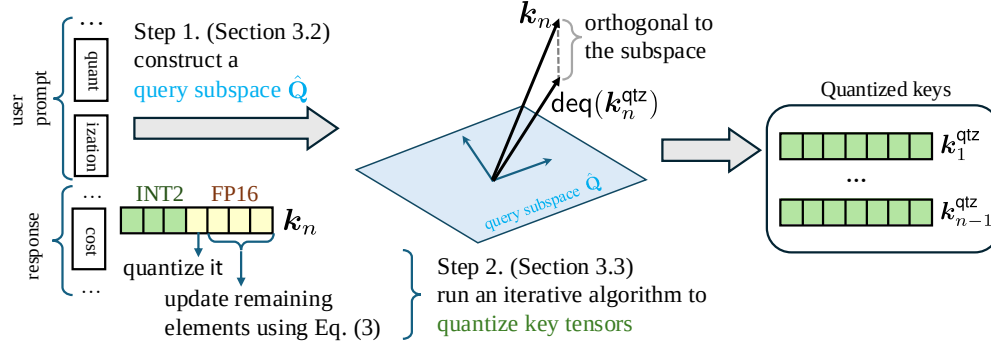


Figure 1: The pipeline of SQuat for quantizing key tensors. We first identify a subspace $\hat{Q}$ spanned by query tensors that carry the most relevant task information necessary for answering the user question. Then we quantize each key tensor k_n element by element (or block by block) and update the remaining elements using Eq. (3) at each iteration. This update ensures that the difference between the (de)quantized and original key remains as orthogonal to the query subspace as possible, preserving critical information for responding to the user question.

3 Main Results

3.1 Motivation and Problem Formulation

Motivation. We first recall how existing methods (e.g., Liu et al., 2024e; Kang et al., 2024; Hooper et al., 2024b) quantize the KV cache. When a new token produces its key tensor, it is not immediately quantized but instead stored in a residual buffer. Once this buffer accumulates R key tensors, they are quantized together. This process involves grouping the key tensors and computing a zero-point and scaling factor for each group. Existing methods explore different strategies to reduce quantization error, such as per-channel grouping and pre-RoPE quantization. Despite these differences, they share the same fundamental goal: minimizing the difference between the (de)quantized and original key tensors: $\|k - \text{deq}(k^{qtz})\|_2$ (with the same principle applied to value tensors). Based on this criterion, we refer to them as *compression-based quantization methods*.

Compression-based methods overlook the impact of quantization on the attention mechanism itself. Since attention scores are determined by the inner products between query and key vectors, even small perturbations in key tensors can lead to disproportionate shifts in attention outputs. The following theorem formalizes this intuition.

Theorem 1. For a new token with query tensor $q_n \in \mathbb{R}^d$, we can upper bound the output of the attention mechanism when using original keys&values compared to quantized keys&values:

$$\begin{aligned} & \left\| \text{Attention}(q_n, \{k_i\}_{i=1}^n, \{v_i\}_{i=1}^n) - \text{Attention}(q_n, \{k_i^{qtz}\}_{i=1}^n, \{v_i^{qtz}\}_{i=1}^n) \right\|_2 \\ & \leq \frac{\sum_{i=1}^n \|v_i - \text{deq}(v_i^{qtz})\|_2}{2\sqrt{d}} \sum_{i=1}^n |q_n(k_i - \text{deq}(k_i^{qtz}))^T| + \sum_{i=1}^n \|v_i - \text{deq}(v_i^{qtz})\|_2. \end{aligned}$$

Theorem 1 indicates that the outputs of the attention mechanism remain unchanged when using quantized tensors, provided two conditions are satisfied: (i) the quantized value tensors closely approximate the original ones, and (ii) the key residuals—the differences between the (de)quantized and original key tensors—are orthogonal to the query tensor. Accordingly, we quantize value tensors on a per-token basis to minimize their approximation errors relative to the original value tensors. For key tensors, we quantize them while ensuring that key residuals remain orthogonal to the query tensors of future tokens.

Problem formulation. We formalize the above intuition for key tensor quantization as an optimization problem. Given a key tensor k , our goal is to find its b -bit representation

k^{qtz} such that the quantization error is minimized, while the residual $k - \text{deq}(k^{\text{qtz}})$ remains orthogonal to the subspace spanned by the query tensors of future tokens—denoted by $\hat{\mathbf{Q}}$ —to preserve task-relevant information. This leads to the following constrained optimization:

$$\min_{k^{\text{qtz}}} \|k - \text{deq}(k^{\text{qtz}})\|_2, \quad \hat{\mathbf{Q}}(k - \text{deq}(k^{\text{qtz}})) = \mathbf{0}, \quad k^{\text{qtz}} \text{ is a vector of } b\text{-bit integers.} \quad (1)$$

We face two challenges to solve this optimization. First, the query tensors of future tokens—and thus the subspace $\hat{\mathbf{Q}}$ —are unavailable when quantizing the current token’s key tensor. In Section 3.2, we explore how to estimate $\hat{\mathbf{Q}}$ without knowing future tokens. Second, the optimization in (1) is combinatorial and generally intractable. In Section 3.3, we introduce an efficient iterative algorithm to obtain an approximate solution.

3.2 Exploring Query Subspace

Our key observation is that query tensors typically lie within a subspace (up to a certain error) whose dimension is significantly smaller than the hidden dimension d . Importantly, this subspace can be derived directly from the prompt tokens, without requiring access to the response. In other words, even when future tokens are unknown at the time of quantizing the key tensors for the current token, this prompt-derived subspace can still guide the quantization process.

We illustrate this observation using a sequence from the math-500 dataset and generate its response with the Llama-3.1-8B-Instruct model. To precisely define what it means for query tensors to lie within a subspace, we introduce the following definition.

Definition 1. For a query tensor q , we define its deviation from a given subspace $\mathcal{P}$ as $\|q - \text{Proj}_{\mathcal{P}}(q)\|_2$, where $\text{Proj}_{\mathcal{P}}(q)$ denotes the projection of q onto $\mathcal{P}$. When the query tensor is normalized, its deviation from any subspace stays within $[0, 1]$.

We apply singular value decomposition (SVD) to query tensors from: (1) all tokens (prompt + response), (2) only prompt tokens, (3) all tokens from a different math-500 sequence, and (4) all tokens from a sequence in QuALITY (Pang et al., 2021), a multiple-choice QA dataset. In each case, we select the top r singular vectors to form the subspace.

Figure 2 (left) illustrates the deviation of (normalized) query tensors of tokens in the math-500 sequence from the four subspaces. As shown, a 30-dimensional subspace can capture the entire sequence’s query tensors with $\sim 20\%$ deviation error. Moreover, using prompt-only query tensors—available before the decoding phase begins—or query tensors from a different math-500 sequence does not increase the deviation error. The deviation increases slightly, however, when the subspace is spanned using a sequence from QuALITY, a different dataset. Based on these observations, we apply SVD to the query tensors of all prompt tokens. Then we define the task-relevant query subspace as $\hat{\mathbf{Q}} \in \mathbb{R}^{r \times d}$, constructed from the top r singular vectors scaled by their corresponding singular values.

3.3 Key Tensors Quantization

We propose an iterative algorithm to solve (1). In each iteration, we select a coordinate¹ of k , quantize it, and update the remaining coordinates to ensure the residual is orthogonal to the query subspace as much as possible. The process repeats for the remaining coordinates until all coordinates of k are quantized.

Specifically, we use $\hat{k}_t \in \mathbb{R}^d$ to represent the (de)quantized value of k at the t -th iteration and set $\hat{k}_0 = k$. At iteration t , the first $t - 1$ elements of $\hat{k}_t$ remain unchanged from $\hat{k}_{t-1}$, as they have already been quantized. We then quantize the t -th element and update the remaining $d - t$ elements using (3) in the following proposition. The hyperparameter $\lambda \in [0, \infty)$ controls the trade-off between making the (de)quantized key close to its original values and

¹In Appendix A.2, we extend our algorithm to do block-by-block quantization of key tensors, rather than element-by-element. In our experiments, each key tensor is partitioned into no more than 4 blocks, leading to at most 4 iterations.

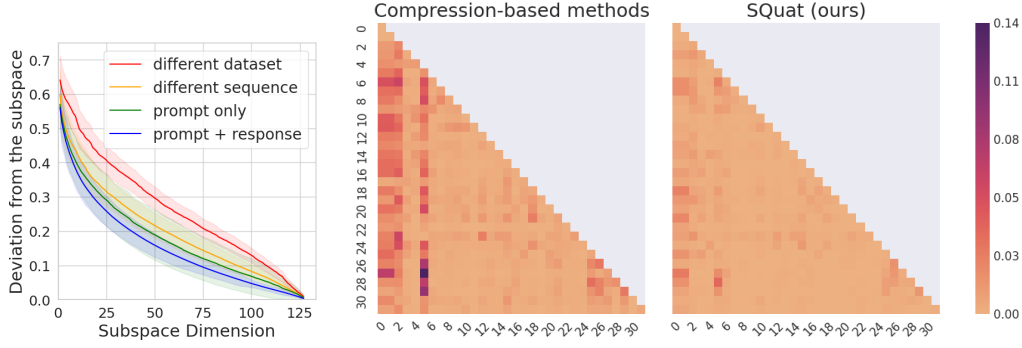


Figure 2: Left: query tensors tend to lie in a low-dimensional subspace. This subspace can be identified using query tensors from prompt tokens, which are available before decoding begins. Middle and Right: the absolute *difference* in attention scores between using original and quantized key tensors. As shown, our method better preserves attention scores. See Appendix D.1 for experimental setup.

enforcing orthogonality of its residuals to the query subspace. When $\lambda = 0$, the algorithm reduces to a compression-based quantization method. For any vector $\mathbf{x}$, we use $[x]_i$ to denote its i -th coordinate.

Proposition 1. Consider the optimization problem

$$\begin{aligned} \min_{\hat{\mathbf{k}}_t \in \mathbb{R}^d} \quad & \|\hat{\mathbf{k}}_t - \hat{\mathbf{k}}_{t-1}\|_2^2 + \lambda \|\hat{\mathbf{Q}}(\hat{\mathbf{k}}_t - \hat{\mathbf{k}}_{t-1})\|_2^2, \\ \text{s.t.} \quad & [\hat{\mathbf{k}}_t]_i = [\hat{\mathbf{k}}_{t-1}]_i \text{ for } i = 1, \dots, t-1 \text{ and } [\hat{\mathbf{k}}_t]_t = \text{deq}(\text{qtz}([\hat{\mathbf{k}}_{t-1}]_t)). \end{aligned} \quad (2)$$

We denote $\mathbf{P}_{\text{inv}} \triangleq (\mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}})^{-1}$. Suppose $\mathbf{A}_t \in \mathbb{R}^{t \times t}$ is the top-left block of $\mathbf{P}_{\text{inv}}$ and $\mathbf{B}_t \in \mathbb{R}^{(d-t) \times t}$ is the bottom left block of $\mathbf{P}_{\text{inv}}$. Let $\mathbf{h}_t$ be the last column of $\mathbf{A}_t^{-1}$. Then the optimal solution to (2) has its last $d - t$ elements given by

$$[\hat{\mathbf{k}}_{t-1}]_t + (\text{deq}(\text{qtz}([\hat{\mathbf{k}}_{t-1}]_t)) - [\hat{\mathbf{k}}_{t-1}]_t) \mathbf{B}_t \mathbf{h}_t. \quad (3)$$

The main computational cost in Proposition 1 arises from computing $\mathbf{A}_t^{-1}$ at each iteration, which has a complexity of $O(\sum_{t=1}^d t^3) = O(d^4)$. The following proposition introduces an iterative algorithm to compute $\mathbf{A}_t^{-1}$. It begins with $\mathbf{A}_d^{-1} = \mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}}$ by definition and subsequently derives $\mathbf{A}_t^{-1}$ from $\mathbf{A}_{t+1}^{-1}$. This way, we reduce the complexity of the iterative algorithm in Proposition 1 from $O(d^4)$ to $O(d^3)$.

Proposition 2. Recall from Proposition 1 that $\mathbf{A}_{t+1} \in \mathbb{R}^{(t+1) \times (t+1)}$ is the top-left block of $\mathbf{P}_{\text{inv}}$.

For $t = d - 1, \dots, 1$, if we write $\mathbf{A}_{t+1}^{-1} = \begin{bmatrix} \bar{\mathbf{A}}_{t+1} & \mathbf{a}_{t+1}^T \\ \mathbf{a}_{t+1} & \bar{a}_{t+1} \end{bmatrix}$, where $\bar{\mathbf{A}}_{t+1} \in \mathbb{R}^{t \times t}$, then we have

$$\mathbf{A}_t^{-1} = \bar{\mathbf{A}}_{t+1} - \frac{1}{\bar{a}_{t+1}} \mathbf{a}_{t+1}^T \mathbf{a}_{t+1}. \quad (4)$$

We show how our method preserves attention scores compared to compression-based methods in Figure 2 (middle and right) and defer a more comprehensive evaluation to the next section. A pseudo-code of our algorithm is provided in Algorithm 1, with an illustration shown in Figure 5.

4 Numerical Experiments

We evaluate SQuat and SQuat^{pre} for 2-bit quantization of the KV cache using four LLMs across 14 benchmark tasks, comparing them with four baseline quantization methods.

Method	KV size	GSM8k	MMLU_Pro_Math	MMLU_Pro_Law	IFEval	GPQA	BBH	Average
Llama-2-7B-hf								
FP16	100%	14.03%	7.25%	16.17%	27.47%	11.61%	39.53%	20.87%
KIVI	19.7%	11.98%	<u>8.51%</u>	12.53%	26.29%	9.60%	<u>35.92%</u>	18.86%
GEAR	22.5%	11.60%	9.25%	13.72%	27.94%	9.60%	35.77%	19.28%
ZipCache	21.6%	6.37%	7.48%	15.80%	25.71%	8.48%	24.73%	15.39%
SQuat	19.7%	13.42%	7.62%	15.35%	<u>27.44%</u>	<u>9.82%</u>	35.76%	19.59%
SQuat^{pre}	19.7%	<u>12.21%</u>	6.66%	13.72%	25.62%	11.83%	36.85%	<u>19.34%</u>
Llama-3.1-8B-Instruct								
FP16	100%	77.56%	39.38%	26.70%	52.96%	16.74%	71.06%	50.27%
KIVI	21.4%	71.49%	30.72%	27.25%	50.80%	13.39%	59.62%	44.86%
GEAR	25.8%	69.37%	30.35%	27.79%	<u>54.32%</u>	15.40%	57.70%	45.17%
ZipCache	22.1%	72.02%	32.72%	<u>27.43%</u>	51.83%	11.16%	55.32%	44.08%
SQuat	21.4%	72.71%	33.83%	27.07%	54.14%	<u>16.96%</u>	60.59%	<u>46.97%</u>
SQuat^{pre}	21.4%	<u>72.56%</u>	<u>32.94%</u>	27.34%	55.91%	17.86%	62.85%	47.86%
Mistral-7B-Instruct-v0.3								
FP16	100%	50.49%	23.09%	24.34%	54.21%	18.30%	56.23%	40.59%
KIVI	20.7%	42.61%	20.21%	23.07%	54.01%	<u>18.53%</u>	47.77%	36.91%
SQuat	20.7%	45.26%	21.24%	<u>23.52%</u>	<u>53.82%</u>	19.87%	<u>50.27%</u>	38.32%
SQuat^{pre}	20.7%	<u>44.50%</u>	22.95%	23.98%	52.28%	<u>18.53%</u>	50.79%	<u>37.91%</u>

Table 1: We compare SQuat and SQuat^{pre} against existing tuning-free KV cache quantization baselines and FP16 (no quantization) across multiple tasks from LM-Eval. KV size is the average ratio of the compressed KV cache to its FP16 counterpart. For each task, the best result is shown in bold, and the second-best is underlined.

The key difference between SQuat and SQuat^{pre} is that SQuat^{pre} quantizes and stores the KV cache *before applying RoPE* (Su et al., 2024), as recommended by Hooper et al. (2024b). Because SQuat^{pre} applies positional embeddings on-the-fly after dequantization, it operates slightly slower than SQuat.

Baselines and LLMs. We consider FP16 (no quantization) and existing KV cache quantization algorithms—KIVI (Liu et al., 2024e), GEAR (Kang et al., 2024), ZipCache (He et al., 2024)—which, like our method, do not require fine-tuning or a calibration dataset. We evaluate these methods on four open-source LLMs: Llama-2-7B, Llama-3.1-8B-Instruct, Mistral-7B-Instruct-v0.3, DeepSeek-R1-Distill-Llama-8B. We assess performance across several reasoning and long-context benchmarks, and report the KV size as the average percentage of the compressed cache relative to the FP16 cache at the end of generation. We present the experimental setup, including the choice of hyper-parameters for SQuat, SQuat^{pre}, and other baseline methods in Appendix D.2.

Reasoning tasks. We evaluate SQuat and baseline methods on a range of reasoning tasks from LM-Eval (Gao et al., 2024), including GSM8k, MMLU_Pro_Math, MMLU_Pro_Law, IFEval, GPQA, and BBH. These benchmarks are widely adopted in prior work on KV cache compression and are known for their difficulty, often requiring multi-step reasoning (e.g., chain-of-thought) before arriving at the final answer. In such settings, KV cache quantization plays a critical role by substantially reducing memory consumption. We present the results in Table 1. As shown, SQuat achieves higher average scores across all models. Additionally, on GSM8k, GPQA, and BBH—three particularly challenging reasoning tasks—our method consistently outperforms all baselines by a significant margin.

LongBench tasks. We evaluate SQuat and baseline methods on long-context benchmarks, following the recommendations of Liu et al. (2024e) and using the same set of tasks (Qasper, QMSum, MultiNews, TREC, TriviaQA, SAMSum, LCC, and RepoBench-P) from LongBench (Bai et al., 2023). These tasks evaluate the ability of LLMs to process and reason over long contexts, such as document-level question answering and summarization, and require deep understanding across diverse real-world scenarios. As shown in Table 2, SQuat achieves higher average

Method	KV size	Qasper	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RepoBench-P	Average
Llama-2-7B-hf										
FP16	100%	9.61%	21.15%	3.51%	66.00%	87.72%	41.53%	66.66%	59.81%	44.50%
KIVI	19.1%	9.43%	20.71%	0.92%	66.00%	87.42%	42.69%	<u>66.47%</u>	59.83%	44.18%
GEAR	20.7%	9.77%	20.47%	0.96%	66.00%	87.42%	42.42%	66.82%	59.48%	44.17%
ZipCache	19.9%	9.22%	19.65%	3.97%	64.50%	<u>87.48%</u>	40.21%	63.44%	55.99%	43.06%
SQuat	19.1%	10.32%	20.97%	<u>3.48%</u>	66.00%	87.28%	42.27%	66.36%	<u>59.59%</u>	44.53%
SQuat ^{pre}	19.1%	<u>9.94%</u>	20.97%	2.44%	65.50%	88.09%	<u>42.44%</u>	66.29%	59.27%	<u>44.37%</u>
Llama-3.1-8B-Instruct										
FP16	100%	45.00%	23.40%	27.21%	69.50%	91.20%	44.01%	63.45%	55.25%	52.38%
KIVI	19.6%	44.03%	<u>23.86%</u>	26.79%	69.50%	91.85%	43.47%	62.45%	53.04%	51.87%
GEAR	21.4%	43.13%	23.26%	26.79%	69.00%	90.74%	43.21%	61.82%	52.87%	51.35%
ZipCache	21.4%	42.87%	23.24%	26.61%	69.50%	91.66%	43.43%	62.64%	51.86%	51.48%
SQuat	19.6%	<u>44.22%</u>	23.55%	26.78%	69.50%	92.19%	44.19%	<u>62.94%</u>	<u>53.09%</u>	<u>52.06%</u>
SQuat ^{pre}	19.6%	44.25%	23.87%	26.78%	69.50%	<u>92.02%</u>	<u>43.88%</u>	63.00%	53.53%	52.10%
Mistral-7B-Instruct-v0.3										
FP16	100%	40.54%	24.34%	27.83%	74.00%	88.39%	47.74%	58.39%	56.89%	52.27%
KIVI	19.2%	38.87%	<u>23.90%</u>	26.94%	74.00%	<u>88.34%</u>	47.74%	57.64%	<u>55.81%</u>	<u>51.66%</u>
SQuat	19.2%	39.57%	23.92%	<u>27.01%</u>	74.00%	88.89%	<u>46.94%</u>	57.82%	55.77%	51.74%
SQuat ^{pre}	19.2%	<u>39.04%</u>	23.80%	27.31%	74.00%	87.94%	46.93%	<u>57.67%</u>	56.42%	51.64%

Table 2: Comparison of different KV cache quantization methods on tasks from LongBench. For each task, the best result is shown in bold, and the second-best is underlined.

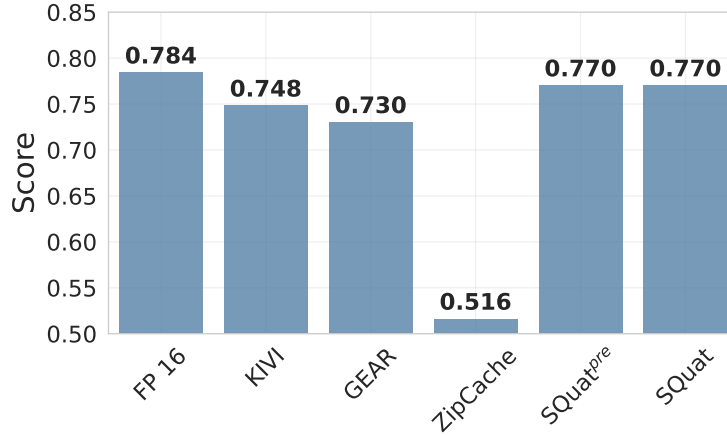


Figure 3: Experiments using a reasoning model (DeepSeek-R1-Distill-Llama-8B) on the math-500 dataset. We compare SQuat and SQuat^{pre} with different KV cache quantization baselines and FP16 (no quantization).

scores across all models than other quantization baselines and are nearly lossless compared with FP16.

Reasoning model. We evaluate DeepSeek-R1-Distill-Llama-8B on the math-500 dataset (Lightman et al., 2023). We set the maximum number of new tokens to 8k to ensure the model has sufficient window for multi-stage reasoning. This setup highlights a key use case for KV cache quantization: modern reasoning models often require long responses to “think” through problems—e.g., by generating chain-of-thought reasoning—before arriving at a final answer. In such scenarios, the KV cache becomes a memory bottleneck, and quantization proves especially helpful when compute resources are limited. As shown in Figure 3, both SQuat and SQuat^{pre} outperform all existing KV cache quantization baselines.

Efficiency comparisons. We apply SQuat to quantize the KV cache to 2-bit and evaluate its efficiency in terms of memory usage, latency, and throughput on the Llama-2-7B model. Empirically, GEAR and ZipCache run much slower than KIVI (Liu et al., 2024e) and SQuat, so we compare SQuat only against KIVI and an FP16 baseline (without quantization) in

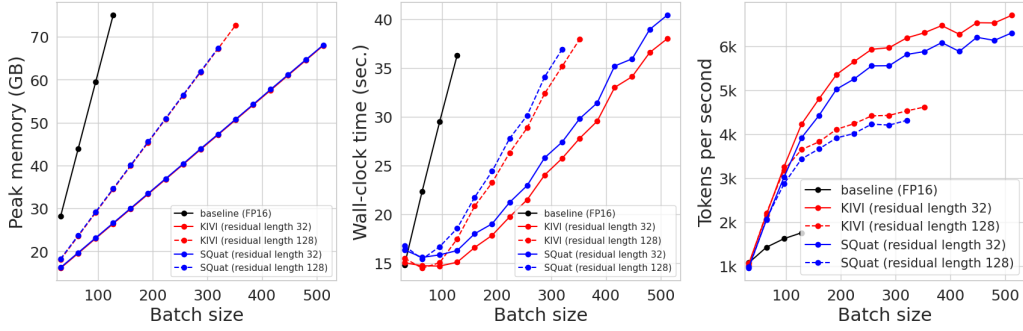


Figure 4: We evaluate SQuat’s efficiency by measuring peak memory usage (left), latency (middle), and throughput (right), comparing it against KIVI (Liu et al., 2024e) and an FP16 baseline. As shown, SQuat requires the same peak GPU memory as KIVI. In terms of throughput, SQuat achieves a $2.45\times \sim 3.60\times$ improvement over FP16, which is comparable to KIVI’s $2.63\times \sim 3.82\times$ improvement.

Figure 4. To ensure a fair comparison, we adopt the same experimental setup as KIVI, which synthesizes workloads based on ShareGPT (2023). We evaluate two residual buffer lengths (i.e., the number of most recent tokens whose KV cache is stored in full precision), 32 and 128, for both our algorithm and KIVI. We gradually increase the batch size until the GPU runs out of memory. All experiments are conducted on a single Nvidia H100 GPU (80GB). As shown in Figure 4, SQuat reduces peak GPU memory by $2.17\times \sim 2.82\times$. This memory efficiency enables up to a $4\times$ increase in batch size compared to the FP16 baseline, leading to a throughput improvement of $2.45\times \sim 3.60\times$, which is comparable to KIVI’s throughput improvement of $2.63\times \sim 3.82\times$.

5 Conclusion, Future work, and Limitations

The past years have seen rapid advancements in techniques for developing LLMs. As SOTA models continue to grow in size to achieve higher performance (Zhao et al., 2023), ensuring accessibility to these techniques—especially for those with limited computational resources—has become increasingly important. This paper responds to this call by exploring methods to make LLM inference-time decoding more efficient. We focus on a specific aspect of this challenge: KV cache quantization. Unlike existing compression-based algorithms, our method quantizes key tensors with the objective of preserving their inner product with future tokens’ query tensors. This design choice ensures that quantization errors minimally affect model outputs. We establish rigorous theoretical foundations for our algorithm and hope our work inspires new research that applies theoretical insights to practical LLM development, promoting efficient and accessible inference techniques.

There are several promising directions for further exploration. First, some recent LLMs do not store the KV cache directly. For instance, multi-head latent attention (Liu et al., 2024a) caches compressed latent vectors and applies projection matrices to generate key and value tensors during inference. Investigating whether these latent vectors can be further quantized, and understanding the impact of quantization errors on model performance, is an interesting avenue for research. From a theoretical standpoint, it is also valuable to examine how reducing KV cache size—through quantization or pruning—affects latency, memory usage, and throughput. Identifying the optimal trade-offs between compression rate and model performance for specific tasks could offer insights into more efficient model deployment strategies.

Acknowledgment

We would like to thank Shivchander Sudalairaj, Abhishek Bhandwaladar, Mustafa Eyceoz, Aldo Pareja from the Red Hat AI Innovation Team for their help in setting up some of the evaluation benchmarks.

References

- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.
- Wael Alghamdi, Hsiang Hsu, Haewon Jeong, Hao Wang, P Winston Michalak, Shahab Asoodeh, and Flavio P Calmon. Beyond adult and compas: Fairness in multi-class prediction. *arXiv preprint arXiv:2206.07801*, 2022.
- Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefer, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37: 100213–100240, 2024.
- Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.
- Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- AH Bentbib and A Kanber. Block power method for svd decomposition. *Analele Stiintifice ale Universitatii Ovidius Constanta, Seria Matematica*, 23(2):45–58, 2015.
- Woohong Byun, Jongseok Woo, and Saibal Mukhopadhyay. Hessian-aware kv cache quantization for llms. In *2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 243–247. IEEE, 2024.
- Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.
- Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.
- Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.
- Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.
- Haojie Duanmu, Zhihang Yuan, Xiuhong Li, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. Skvq: Sliding-window key and value cache quantization for large language models. *arXiv preprint arXiv:2405.06219*, 2024.

- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488, 2022.
- Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.
- Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac’h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2024. URL <https://zenodo.org/records/12608602>.
- Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pp. 1737–1746. PMLR, 2015.
- Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*, 2023.
- Babak Hassibi, David Stork, and Gregory Wolff. Optimal brain surgeon: Extensions and performance comparisons. *Advances in neural information processing systems*, 6, 1993.
- Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification. *Advances in Neural Information Processing Systems*, 37:68287–68307, 2024.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, June Paik, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length llm inference. *arXiv preprint arXiv:2411.09688*, 2024a.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024b.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. Mistral 7b, 2023.

- Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *Advances in Neural Information Processing Systems*, 37:52481–52515, 2024.
- Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. S3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.
- Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*, 2024.
- Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.
- Junhyuck Kim, Jongho Park, Jaewoong Cho, and Dimitris Papailiopoulos. Lexico: Extreme kv cache compression via sparse coding over universal dictionaries. *arXiv preprint arXiv:2412.08890*, 2024.
- Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 784–794, 2022.
- Eldar Kurtić, Elias Frantar, and Dan Alistarh. Ziplm: Inference-aware structured pruning of language models. *Advances in Neural Information Processing Systems*, 36:65597–65617, 2023.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with paged attention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.
- Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.
- Heejun Lee, Geon Park, Jaduk Suh, and Sung Ju Hwang. Infinitehip: Extending language model context up to 3 million tokens on a single gpu. *arXiv preprint arXiv:2502.08910*, 2025.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.
- Pierre Lienhart. Llm inference series: 4. kv caching, a deeper look. <https://medium.com/@plienhar/llm-inference-series-4-kv-caching-a-deeper-look-4ba9a77746c8>, 2024.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.
- Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.
- Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *arXiv preprint arXiv:2405.14366*, 2024b.
- Ruikang Liu, Haoli Bai, Haokun Lin, Yuening Li, Han Gao, Zhengzhuo Xu, Lu Hou, Jun Yao, and Chun Yuan. Intactkv: Improving large language model quantization by keeping pivot tokens intact. *arXiv preprint arXiv:2403.01241*, 2024c.

- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36, 2024d.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024e.
- Vladimir Malinovskii, Andrei Panferov, Ivan Ilin, Han Guo, Peter Richtárik, and Dan Alistarh. Pushing the limits of large language model quantization via the linearity theorem. *arXiv preprint arXiv:2411.17525*, 2024.
- NVIDIA. Fastertransformer. <https://github.com/NVIDIA/FasterTransformer>, 2023.
- OpenAI. Introducing openai o1. <https://openai.com/o1/>, 2024.
- Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, et al. Quality: Question answering with long input texts, yes! *arXiv preprint arXiv:2112.08608*, 2021.
- Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference. *arXiv preprint arXiv:2312.04985*, 2023.
- Sasha Rush and Daniel Ritter. Speculations on test-time scaling. <https://github.com/srush/awesome-o1?tab=readme-ov-file#ref-Lightman2023-cr>, 2024.
- ShareGPT. Sharegpt. <https://sharegpt.com>, 2023.
- Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.
- Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.
- Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.
- Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.
- Qian Tao, Wenyuan Yu, and Jingren Zhou. Asymkv: Enabling 1-bit quantization of kv cache with layer-wise asymmetric quantization configurations. *arXiv preprint arXiv:2410.13212*, 2024.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- Raushan Turganbay. Unlocking longer generation with key-value cache quantization. <https://huggingface.co/blog/kv-cache-quantization>, 2024.
- Shashank Verma and Neal Vaidya. Mastering llm techniques: Inference optimization. <https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization>, 2023.

- Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, and Mi Zhang. D2o: Dynamic discriminative operations for efficient generative inference of large language models. *arXiv preprint arXiv:2406.13035*, 2024.
- Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*, 2024.
- Haoyi Wu and Kewei Tu. Layer-condensed kv cache for efficient inference of large language models. *arXiv preprint arXiv:2405.10637*, 2024.
- Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023a.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023b.
- Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Think: Thinner key cache by query-driven pruning. *arXiv preprint arXiv:2407.21018*, 2024.
- Zhen Yang, JN Han, Kan Wu, Ruobing Xie, An Wang, Xingwu Sun, and Zhanhui Kang. Lossless kv cache compression to 2%. *arXiv preprint arXiv:2410.15252*, 2024.
- Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems*, 35:27168–27183, 2022.
- Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.
- Hao Yu, Zelan Yang, Shen Li, Yong Li, and Jianxin Wu. Effectively compress kv heads for llm. *arXiv preprint arXiv:2406.07056*, 2024.
- Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. *arXiv preprint arXiv:2502.11089*, 2025.
- Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*, 2024.
- Amir Zandieh, Majid Daliri, and Insu Han. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead. *arXiv preprint arXiv:2406.03482*, 2024.
- Hongxuan Zhang, Yao Zhao, Jiaqi Zheng, Chenyi Zhuang, Jinjie Gu, and Guihai Chen. Csr: Achieving 1 bit key-value cache via sparse representation. *arXiv preprint arXiv:2412.11741*, 2024a.
- Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient llms inference. In *Forty-first International Conference on Machine Learning*, 2024b.
- Zeyu Zhang and Haiying Shen. Zero-delay qkv compression for mitigating kv cache and network bottlenecks in llm inference. *arXiv preprint arXiv:2408.04107*, 2024.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.

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

A Omitted Proofs

A.1 Proof of Theorem 1

Proof. We denote $\mathbf{k}_i^{\text{deq}} \triangleq \text{deq}(\mathbf{k}_i^{\text{qtz}})$ and $\mathbf{v}_i^{\text{deq}} \triangleq \text{deq}(\mathbf{v}_i^{\text{qtz}})$ for $i = 1, \dots, n$. We introduce two vectors:

$$\mathbf{w} = \left[\frac{\langle \mathbf{q}_n, \mathbf{k}_1 \rangle}{\sqrt{d}}, \dots, \frac{\langle \mathbf{q}_n, \mathbf{k}_n \rangle}{\sqrt{d}} \right], \quad \mathbf{w}^{\text{deq}} = \left[\frac{\langle \mathbf{q}_n, \mathbf{k}_1^{\text{deq}} \rangle}{\sqrt{d}}, \dots, \frac{\langle \mathbf{q}_n, \mathbf{k}_n^{\text{deq}} \rangle}{\sqrt{d}} \right].$$

Let $\mathbf{V}, \mathbf{V}^{\text{deq}} \in \mathbb{R}^{n \times d}$, where the rows consist of $\mathbf{v}_i$ and $\mathbf{v}_i^{\text{deq}}$, respectively. Then we have

$$\begin{aligned} & \left\| \text{Attention}(\mathbf{q}_n, \{\mathbf{k}_i\}_{i=1}^n, \{\mathbf{v}_i\}_{i=1}^n) - \text{Attention}(\mathbf{q}_n, \{\mathbf{k}_i^{\text{qtz}}\}_{i=1}^n, \{\mathbf{v}_i^{\text{qtz}}\}_{i=1}^n) \right\|_2 \\ &= \left\| \text{softmax}(\mathbf{w})\mathbf{V} - \text{softmax}(\mathbf{w}^{\text{deq}})\mathbf{V}^{\text{deq}} \right\|_2 \\ &= \left\| \text{softmax}(\mathbf{w})(\mathbf{V} - \mathbf{V}^{\text{deq}}) + (\text{softmax}(\mathbf{w}) - \text{softmax}(\mathbf{w}^{\text{deq}}))\mathbf{V}^{\text{deq}} \right\|_2 \\ &\leq \left\| \text{softmax}(\mathbf{w})(\mathbf{V} - \mathbf{V}^{\text{deq}}) \right\|_2 + \left\| (\text{softmax}(\mathbf{w}) - \text{softmax}(\mathbf{w}^{\text{deq}}))\mathbf{V}^{\text{deq}} \right\|_2. \end{aligned}$$

Note that

$$\left\| \text{softmax}(\mathbf{w})(\mathbf{V} - \mathbf{V}^{\text{deq}}) \right\|_2 \leq \sum_{i=1}^n \left\| \mathbf{v}_i - \mathbf{v}_i^{\text{deq}} \right\|_2.$$

Similarly, we have

$$\left\| (\text{softmax}(\mathbf{w}) - \text{softmax}(\mathbf{w}^{\text{deq}}))\mathbf{V}^{\text{deq}} \right\|_2 \leq \left\| \text{softmax}(\mathbf{w}) - \text{softmax}(\mathbf{w}^{\text{deq}}) \right\|_2 \left\| \mathbf{V}^{\text{deq}} \right\|_F.$$

Note that softmax is 1/2-Lipschitz continuous (see Appendix A.4 in [Alghamdi et al., 2022](#)). Therefore,

$$\begin{aligned} \left\| \text{softmax}(\mathbf{w}) - \text{softmax}(\mathbf{w}^{\text{deq}}) \right\|_2 &\leq \frac{1}{2} \left\| \mathbf{w} - \mathbf{w}^{\text{deq}} \right\|_2 \\ &\leq \frac{1}{2\sqrt{d}} \sqrt{\langle \mathbf{q}_n, (\mathbf{k}_1 - \mathbf{k}_1^{\text{deq}}) \rangle^2 + \dots + \langle \mathbf{q}_n, (\mathbf{k}_n - \mathbf{k}_n^{\text{deq}}) \rangle^2} \\ &\leq \frac{1}{2\sqrt{d}} \sum_{i=1}^n |\mathbf{q}_n(\mathbf{k}_i - \mathbf{k}_i^{\text{deq}})^T|. \end{aligned}$$

□

A.2 Block-Wise Quantization

The algorithm presented in the main body quantizes the key vectors element by element. In this section, we extend it to perform block-wise quantization, improving efficiency by quantizing multiple elements of the key vectors simultaneously while updating the remaining elements. As special cases of this approach, we prove Proposition 1 and Proposition 2.

Before diving into the details, we first recall some standard results from linear algebra that will be used in our proofs.

Lemma 1. Consider a block matrix:

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix},$$

where $\mathbf{A}$ and $\mathbf{D}$ are square matrices. We denote $\mathbf{M}/\mathbf{D} \triangleq \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C}$ and $\mathbf{M}/\mathbf{A} \triangleq \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}$. Suppose $\mathbf{D}$ and $\mathbf{M}/\mathbf{D}$ are both invertible. Then

$$\mathbf{M}^{-1} = \begin{bmatrix} (\mathbf{M}/\mathbf{D})^{-1} & -(\mathbf{M}/\mathbf{D})^{-1}\mathbf{B}\mathbf{D}^{-1} \\ -\mathbf{D}^{-1}\mathbf{C}(\mathbf{M}/\mathbf{D})^{-1} & \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{C}(\mathbf{M}/\mathbf{D})^{-1}\mathbf{B}\mathbf{D}^{-1} \end{bmatrix}.$$

Similarly, suppose $\mathbf{A}$ and $\mathbf{M}/\mathbf{A}$ are both invertible. Then

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1} \\ -(\mathbf{M}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1} & (\mathbf{M}/\mathbf{A})^{-1} \end{bmatrix}.$$

Using this lemma, we immediately obtain the following corollary, which is also a standard result in linear algebra.

Corollary 1. We denote

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}, \quad \mathbf{M}^{-1} = \begin{bmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{bmatrix}.$$

Suppose $\mathbf{A}$ and $\mathbf{M}/\mathbf{A}$ are both invertible. Then we have $\mathbf{A}^{-1} = \mathbf{P} - \mathbf{Q}\mathbf{S}^{-1}\mathbf{R}$.

Proof. Lemma 1 implies that

$$\begin{aligned} \mathbf{P} &= \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1}, \\ \mathbf{Q} &= -\mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1}, \\ \mathbf{R} &= -(\mathbf{M}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1}, \\ \mathbf{S} &= (\mathbf{M}/\mathbf{A})^{-1}. \end{aligned}$$

Substituting the last three equations into the first one yields

$$\mathbf{A}^{-1} = \mathbf{P} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{M}/\mathbf{A})^{-1}\mathbf{C}\mathbf{A}^{-1} = \mathbf{P} - \mathbf{Q}\mathbf{S}^{-1}\mathbf{R}.$$

□

Now, we consider quantizing a block of g elements per iteration in a sequential manner. Suppose the hidden dimension d of the key tensors is divisible by g . At iteration t , the first $(t-1)g$ elements have already been quantized and are kept fixed. During this step, we quantize the elements indexed from $(t-1)g+1$ to tg and update the rest of the elements by using the following lemma.

Lemma 2. Consider the optimization problem

$$\begin{aligned} \min_{\hat{\mathbf{k}}_t \in \mathbb{R}^d} \quad & \|\hat{\mathbf{k}}_t - \hat{\mathbf{k}}_{t-1}\|_2^2 + \lambda \|\hat{\mathbf{Q}}(\hat{\mathbf{k}}_t - \hat{\mathbf{k}}_{t-1})\|_2^2, \\ \text{s.t.} \quad & [\hat{\mathbf{k}}_t]_i = [\hat{\mathbf{k}}_{t-1}]_i \quad \text{for } i = 1, \dots, (t-1)g \\ & [\hat{\mathbf{k}}_t]_i = \text{deq}(\text{qtz}([\hat{\mathbf{k}}_{t-1}]_i)) \quad \text{for } i = (t-1)g+1, \dots, tg. \end{aligned} \tag{5}$$

We denote

$$\mathbf{P}_{inv} \triangleq (\mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}})^{-1} = \begin{bmatrix} \mathbf{A}_t & \mathbf{B}_t^T \\ \mathbf{B}_t & \mathbf{C}_t \end{bmatrix},$$

where $\mathbf{A}_t \in \mathbb{R}^{tg \times tg}$ is the top-left block of $\mathbf{P}_{inv}$. Let $\mathbf{H}_t$ be the last g columns of $\mathbf{A}_t^{-1}$. The optimization in (5) has a closed-form optimal solution:

$$[\hat{\mathbf{k}}_t]_i = \begin{cases} [\hat{\mathbf{k}}_{t-1}]_i & i = 1, \dots, (t-1)g \\ \text{deq}(\text{qtz}([\hat{\mathbf{k}}_{t-1}]_i)) & i = (t-1)g+1, \dots, tg \end{cases}$$

and the remaining elements of $\hat{\mathbf{k}}_t$ are given by

$$[\hat{\mathbf{k}}_{t-1}]_{tg:} + \mathbf{B}_t \mathbf{H}_t \mathbf{d}, \quad (6)$$

where $\mathbf{d} \in \mathbb{R}^g$ has its i -th element corresponding to the $(t-1)g + i$ -th element of $\text{deq}(\text{qtz}(\hat{\mathbf{k}}_{t-1})) - \hat{\mathbf{k}}_{t-1}$.

Proof. We denote $\boldsymbol{\delta} = \hat{\mathbf{k}}_t - \hat{\mathbf{k}}_{t-1}$ and let $\mathbf{d} \in \mathbb{R}^g$ has its i -th element corresponding to the $(t-1)g + i$ -th element of $\text{deq}(\text{qtz}(\hat{\mathbf{k}}_{t-1})) - \hat{\mathbf{k}}_{t-1}$. Moreover, we denote

$$\mathbf{T} \triangleq [\mathbf{I}, \mathbf{0}], \quad \mathbf{b} \triangleq \begin{bmatrix} \mathbf{0} \\ \mathbf{d} \end{bmatrix},$$

where the identity matrix has dimension $tg \times tg$ and $\mathbf{b} \in \mathbb{R}^{tg}$. The optimization problem in (5) can be rewritten as

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^d} \boldsymbol{\delta}^T \boldsymbol{\delta} + \lambda \boldsymbol{\delta}^T \hat{\mathbf{Q}}^T \hat{\mathbf{Q}} \boldsymbol{\delta}, \quad \mathbf{T} \boldsymbol{\delta} = \mathbf{b}.$$

This expression can be simplified to

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^d} \boldsymbol{\delta}^T \mathbf{P} \boldsymbol{\delta}, \quad \mathbf{T} \boldsymbol{\delta} = \mathbf{b}, \quad (7)$$

where we denote $\mathbf{P} \triangleq \mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}}$. Note that $\mathbf{P}$ is positive definite so it is invertible. We denote $\mathbf{P}_{\text{inv}} \triangleq \mathbf{P}^{-1}$. The Lagrangian of (7) is:

$$L(\boldsymbol{\delta}, \boldsymbol{\lambda}) = \boldsymbol{\delta}^T \mathbf{P} \boldsymbol{\delta} - \boldsymbol{\lambda}^T (\mathbf{T} \boldsymbol{\delta} - \mathbf{b}).$$

The KKT conditions give

$$2\mathbf{P} \boldsymbol{\delta} = \mathbf{T}^T \boldsymbol{\lambda}, \quad (8)$$

$$\mathbf{T} \boldsymbol{\delta} = \mathbf{b}. \quad (9)$$

Hence, we have

$$\begin{aligned} \boldsymbol{\delta} &= \frac{1}{2} \mathbf{P}_{\text{inv}} \mathbf{T}^T \boldsymbol{\lambda}, \\ \frac{1}{2} \mathbf{T} \mathbf{P}_{\text{inv}} \mathbf{T}^T \boldsymbol{\lambda} &= \mathbf{b}. \end{aligned}$$

Suppose $\mathbf{T} \mathbf{P}_{\text{inv}} \mathbf{T}^T$ is invertible. Then we have

$$\boldsymbol{\lambda} = 2(\mathbf{T} \mathbf{P}_{\text{inv}} \mathbf{T}^T)^{-1} \mathbf{b}.$$

Substituting the above equation into (8) leads to

$$\boldsymbol{\delta} = \mathbf{P}_{\text{inv}} \mathbf{T}^T (\mathbf{T} \mathbf{P}_{\text{inv}} \mathbf{T}^T)^{-1} \mathbf{b}. \quad (10)$$

Recall that $\mathbf{T} = [\mathbf{I}, \mathbf{0}]$ and $\mathbf{P}_{\text{inv}} = \begin{bmatrix} \mathbf{A}_t & \mathbf{B}_t^T \\ \mathbf{B}_t & \mathbf{C}_t \end{bmatrix}$. Hence, we have

$$\mathbf{P}_{\text{inv}} \mathbf{T}^T = \begin{bmatrix} \mathbf{A}_t \\ \mathbf{B}_t \end{bmatrix}, \quad \mathbf{T} \mathbf{P}_{\text{inv}} \mathbf{T}^T = \mathbf{A}_t.$$

As a result, we have

$$\mathbf{P}_{\text{inv}} \mathbf{T}^T (\mathbf{T} \mathbf{P}_{\text{inv}} \mathbf{T}^T)^{-1} = \begin{bmatrix} \mathbf{A}_t \\ \mathbf{B}_t \end{bmatrix} (\mathbf{A}_t)^{-1} = \begin{bmatrix} \mathbf{I} \\ \mathbf{B}_t (\mathbf{A}_t)^{-1} \end{bmatrix}.$$

Substituting the above equation and $\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ \mathbf{d} \end{bmatrix}$ into (10) leads to

$$\boldsymbol{\delta} = \begin{bmatrix} \mathbf{0} \\ \mathbf{d} \\ \mathbf{B}_t (\mathbf{A}_t)^{-1} \mathbf{b} \end{bmatrix}.$$

Recall that $\mathbf{H}_t$ is the last g columns of $(\mathbf{A}_t)^{-1}$. Then

$$\delta = \begin{bmatrix} 0 \\ d \\ \mathbf{B}_t \mathbf{H}_t d \end{bmatrix}.$$

□

Now we are in position to prove Proposition 1.

Proof of Proposition 1. Proposition 1 can be viewed as a special case of Lemma 2 by setting $g = 1$. In this case, the vector d becomes a scalar $\text{deq}(\text{qtz}([\hat{k}_{t-1}]_t)) - [\hat{k}_{t-1}]_t$ and the matrix $\mathbf{H}_t$ becomes a vector h_t . □

Lemma 3. Suppose we can write

$$\mathbf{A}_{t+1}^{-1} = \begin{bmatrix} \mathbf{M}_{t+1} & \mathbf{N}_{t+1}^T \\ \mathbf{N}_{t+1} & \mathbf{O}_{t+1} \end{bmatrix},$$

where $\mathbf{M}_{t+1} \in \mathbb{R}^{tg \times tg}$, $\mathbf{N}_{t+1} \in \mathbb{R}^{g \times tg}$, $\mathbf{O}_{t+1} \in \mathbb{R}^{g \times g}$. Then we have

$$\mathbf{A}_t^{-1} = \mathbf{M}_{t+1} - \mathbf{N}_{t+1}^T \mathbf{O}_{t+1}^{-1} \mathbf{N}_{t+1}.$$

Proof. This lemma can be obtained directly by applying Corollary 1. □

Proof of Proposition 2. This proposition is a special case of Lemma 3 and can alternatively be derived using Lemma 1 from Frantar & Alistarh (2022). □

Remark 1. Assume $T = d/g$ is an integer. By definition, $\mathbf{A}_T = \mathbf{P}_{\text{inv}} = (\mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}})^{-1}$, which implies that $\mathbf{A}_T^{-1} = \mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}}$. Then, by using Lemma 3, we can iteratively compute $\mathbf{A}_{T-1}^{-1}, \dots, \mathbf{A}_1^{-1}$.

Proposition 3. The iterative algorithm based on Proposition 1 has a computational complexity of $O(d^4)$. By applying Proposition 2 to update $\mathbf{A}_t^{-1}$, this complexity is reduced to $O(d^3)$.

Proof. The matrix $\mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}}$ has dimensions $d \times d$, and computing its inverse to obtain $\mathbf{P}_{\text{inv}}$ has a complexity² of $O(d^3)$. For $\mathbf{A}_t \in \mathbb{R}^{t \times t}$, directly computing its inverse requires $O(t^3)$ operations. Additionally, computing $\mathbf{B}_t h_t$ involves $O((d-t)t)$ operations. Since the iterative algorithm processes $t = 1, \dots, d$, the total computational cost is:

$$O\left(d^3 + \sum_{t=1}^d t^3 + (d-t)t\right) = O(d^4).$$

By leveraging Proposition 2, the cost of updating $\mathbf{A}_t^{-1}$ reduces to $O(t^2)$ per iteration, decreasing the overall computational complexity to $O(d^3)$. □

B Memory Requirements for KV Cache Management

Maintaining a KV cache requires substantial memory. Specifically, for a batch of b queries, each with a sequence length of $(l_{\text{prompt}} + l_{\text{response}})$, an LLM with L layers, h attention heads per layer, hidden dimension d , and precision p (e.g., 2 bytes per parameter for FP16) requires memory $= 2 \times b \times (l_{\text{prompt}} + l_{\text{response}}) \times L \times h \times d \times p$ bytes for storing the KV cache. In the case of an LLaMA-2 7B model with a batch of 4 queries and a context length of 2048, the memory requirement is $2 \times 4 \times 2048 \times 32 \times 32 \times 128 \times 2$ bytes ≈ 4 GB.

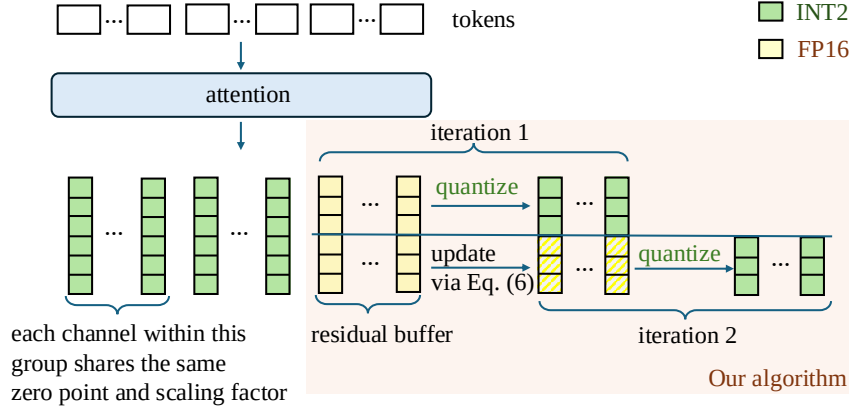


Figure 5: We illustrate how our iterative algorithm quantizes key tensors. We maintain a residual buffer to store the most recent tokens’ key tensors in their original precision. Once the buffer reaches a predefined size (e.g., 32 in our experiments), we apply our quantization algorithm to all stored tensors. Specifically, we first quantize a block of elements (e.g., the first half of the hidden dimensions), then update the remaining elements using Eq. (6). This process continues iteratively until all elements are quantized.

C Details of Our Main Algorithm

We present the pseudocode of our main algorithm in Algorithm 1. As our main focus is on quantizing key tensors, we describe only this process. For value tensors, we adopt a similar procedure with Algorithm 1 in Liu et al. (2024e). Specifically, we maintain a residual buffer that stores the most recent tokens’ value tensors in their original data type (e.g., FP16). When the residual buffer reaches a size of $R + 1$, we quantize the oldest token’s value tensor and remove it from the buffer. Value tensors are quantized per token, with each token’s hidden dimension d divided into $\lceil d/G \rceil$ groups of size G . For the sake of illustration, we assume the hidden dimension d is divisible by the number of elements quantized per iteration g the group size G ; and that the residual buffer length R is divisible by G .

Figure 5 illustrates how our algorithm operates. Once the residual buffer reaches a predefined size, we apply our quantization algorithm to all stored tensors. We begin by quantizing a block of elements, then update the remaining elements using Eq. (6), repeating this process until all elements are quantized. Compared to Eq. (3), Eq. (6) is a more general formulation that supports block-wise quantization rather than quantizing one element at a time. For illustration, the figure assumes that the residual length equals the group size, though our method allows for setting the group size smaller than the residual length to enable finer-grained quantization.

In the pseudocode, we present our algorithm using multi-head attention. However, the method naturally extends to grouped-query attention. For example, our experiments with Llama-3.1-8B-Instruct—which uses grouped-query attention with 8 KV heads and 32 attention heads—follow the same procedure. The only modification is in Line 5: for each KV head, we concatenate the query tensors from its associated query heads before applying SVD.

We assume a batch size of 1 in the pseudocode. When the batch size exceeds 1, we share the same matrices $\mathbf{A}_1^{-1}, \dots, \mathbf{A}_{T-1}^{-1}, \mathbf{P}_{\text{inv}}$ across all samples in the batch, computing them from the first sample. This significantly reduces memory usage for large batch sizes. Empirically, we find that this does not degrade LLM performance compared to computing separate matrices for each sample. This aligns with our observation in Section 3.2 that query tensors tend to

²If r is significantly smaller than d , the Woodbury identity can be applied to reduce this complexity to $O(rd^2)$.

lie within a small subspace, even when constructed from different sequences in the same dataset.

Algorithm 1 Main Algorithm for Key Tensor Quantization

```

1: Input: Residual length  $R$ , group size  $G$ , number of elements to quantize per iteration  $g$ , hidden
   state dimension  $d$ , hyper-parameter  $\lambda$ , subspace dimension  $r$ 

2: // Prefill Phase
3: Input:  $\mathbf{X} \in \mathbb{R}^{l_{\text{prompt}} \times d}$ 
4:  $\mathbf{Q} = \mathbf{XW}^Q$ ,  $\mathbf{K} = \mathbf{XW}^K$ 
5:  $\Sigma, \mathbf{V} = \text{SVD}(\mathbf{Q})$   $\triangleright$  can be accelerated using block power method (Bentbib & Kanber, 2015)
6:  $\hat{\mathbf{Q}} = \text{diag}(\Sigma[:r])\mathbf{V}[:r]$ 
7:  $T = d/g$   $\triangleright$  total iterations required to run our algorithm
8: Compute  $\mathbf{A}_T^{-1} = \mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}}$  and  $\mathbf{P}_{\text{inv}} = (\mathbf{I} + \lambda \hat{\mathbf{Q}}^T \hat{\mathbf{Q}})^{-1}$ 
9: Apply Lemma 3 to obtain  $\mathbf{A}_{T-1}^{-1}, \dots, \mathbf{A}_1^{-1}$ 
10:  $r = l_{\text{prompt}} \% R$ ,  $\text{num\_group} = (l_{\text{prompt}} - r)/G$ 
11: Initialize  $\mathbf{K}^{\text{quant}} = []$  and set  $\mathbf{K}^{\text{full}} = \mathbf{K}[l_{\text{prompt}} - r : ]$ 
12: // Quantize each group of keys (run in parallel)
13: for  $i = 1$  to  $\text{num\_group}$  do
14:    $\mathbf{K}_g = \mathbf{K}[(i-1)G + 1 : iG]$   $\triangleright$  elements in the  $i$ -th group to be quantized
15:    $\mathbf{K}_g^{\text{quant}} \leftarrow \text{QUANT}(\mathbf{K}_g, \mathbf{A}_1^{-1}, \dots, \mathbf{A}_{T-1}^{-1}, \mathbf{P}_{\text{inv}})$ 
16:   Append  $\mathbf{K}_g^{\text{quant}}$  to  $\mathbf{K}^{\text{quant}}$ 
17: end for

18: // Decoding Phase
19: Input:  $\mathbf{x} \in \mathbb{R}^{1 \times d}$ 
20:  $\mathbf{k} = \mathbf{xW}^K$  and Append  $\mathbf{k}$  to  $\mathbf{K}^{\text{full}}$ 
21:  $\text{num\_group} = R/G$ 
22: if  $\text{len}(\mathbf{K}^{\text{full}}) = R$  then
23:   for  $i = 1$  to  $\text{num\_group}$  do
24:      $\mathbf{K}_g = \mathbf{K}^{\text{full}}[(i-1)G + 1 : iG]$ 
25:      $\mathbf{K}_g^{\text{quant}} \leftarrow \text{QUANT}(\mathbf{K}_g, \mathbf{A}_1^{-1}, \dots, \mathbf{A}_{T-1}^{-1}, \mathbf{P}_{\text{inv}})$ 
26:     Append  $\mathbf{K}_g^{\text{quant}}$  to  $\mathbf{K}^{\text{quant}}$ 
27:   end for
28:   Reset  $\mathbf{K}^{\text{full}} = []$ 
29: end if

30: function  $\text{QUANT}(\mathbf{K}, \mathbf{A}_1^{-1}, \dots, \mathbf{A}_{T-1}^{-1}, \mathbf{P}_{\text{inv}})$ 
31:   for  $t = 1$  to  $T$  do
32:     Quantize  $\mathbf{K}[:, (t-1)g + 1 : tg]$  per channel
33:     if  $t < T$  then
34:       Update  $\mathbf{K}[:, tg : ]$  using Lemma 2 with  $\mathbf{A}_t^{-1}$  and  $\mathbf{P}_{\text{inv}}$ 
35:     end if
36:   end for
37:   Return: Quantized  $\mathbf{K}$ 
38: end function

```

D More Experimental Results

D.1 Experimental Setup for Figure 2

We describe the experimental setup for the results presented in Figure 2. Unlike the experiments in Section 4, Figure 2 is based on a semi-synthetic setup. Specifically, we first use the Llama-3.1-8B-Instruct model to generate responses *without* KV cache quantization during decoding, ensuring that quantization errors do not alter the response trajectory. Additionally, we analyze query and key tensors before applying rotary position embeddings (RoPE) (Su et al., 2024) to isolate quantization effects from positional encoding. Figure 2 (left)

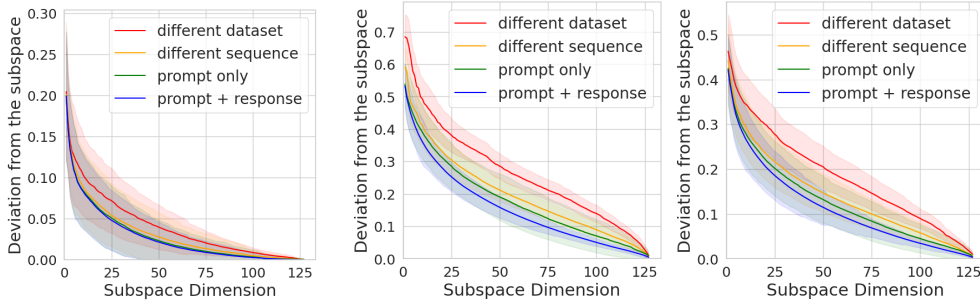


Figure 6: We reproduce the results from Figure 2 (left) for query tensors at different attention layers: the 1st layer (left), the 24th layer (middle), and the 32nd layer (right).

shows the deviation error for query tensors at the 12th attention layer. To further validate our findings, Figure 6 extends these results across different layers.

D.2 Experimental Setup for Section 4

For SQuat, SQuat^{pre}, we set the group size (i.e., the number of elements quantized together) to 32 and the residual length (i.e., the number of most recent tokens whose KV cache is stored in full precision) to 32. The KV cache is quantized to 2 bits. This configuration provides the best throughput and minimal memory usage (see Figure 4). For LM-Eval tasks and all LLMs, we set the subspace dimension to $r = 5$, the regularization coefficient to $\lambda = 0.001$, and the number of elements to quantize per iteration to $g = 64$, resulting in just two iterations of our algorithm. For LongBench tasks, which involve longer inputs, we adjust the hyper-parameters as follows. For Llama-2-7B-hf, we use $r = 5$, $\lambda = 0.001$, $g = 32$ for SQuat^{pre}, and $r = 40$, $\lambda = 0.001$, $g = 64$ for SQuat. For Llama-3.1-8B-Instruct, we set SQuat^{pre} to $r = 60$, $\lambda = 0.0005$, $g = 32$, and SQuat to $r = 10$, $\lambda = 0.0005$, $g = 64$. For Mistral-7B-Instruct-v0.3, we use $r = 20$, $\lambda = 0.0005$, $g = 64$ for SQuat^{pre}, and $r = 5$, $\lambda = 0.001$, $g = 64$ for SQuat. For our experiments using DeepSeek-R1-Distill-Llama-8B on the math-500 dataset (Figure 3), we set $r = 40$, $\lambda = 0.0005$, and $g = 64$ for both SQuat and SQuat^{pre}.

For KIVI and GEAR, we use the same configuration: a group size of 32, a residual length of 32, and 2-bit quantization for the KV cache. For KIVI, we use the official implementation from their GitHub [repository](#). For GEAR, we set $\text{rank} = \text{rankv} = 2$ and $\text{loop} = 3$, following the recommendations in their test script. Since their CUDA-based implementation does not support the Mistral model, we compare only on LLaMA models using their implementation from GitHub [repository](#). For ZipCache, we quantize important tokens to 4 bits and unimportant tokens to 2 bits, with $\text{streaming_gap} = 32$ to match the setup used for other baselines and our method. We set the unimportant_ratio to 0.6 for Llama-2-7B on LM-eval tasks, 0.7 on LongBench tasks, 0.5 for Llama-3.1-8B-Instruct on LM-eval tasks, and 0.6 on LongBench tasks. These values ensure that their KV cache size is roughly aligned with that of other baselines, allowing for a fair comparison. As ZipCache is only implemented for LLaMA in their GitHub [repository](#) (as of the date of our paper submission), we include it in comparisons for LLaMA models only.

For LM-Eval, since MMLU_Pro_Math and MMLU_Pro_Law all come from MMLU_Pro, we compute the weighted average in the follow way:

$$\text{Average} = \frac{1}{5} \text{GSM8k} + \frac{1}{10} (\text{MMLU_Pro_Math} + \text{MMLU_Pro_Law}) + \frac{1}{5} \text{IFEval} + \frac{1}{5} \text{GPQA} + \frac{1}{5} \text{BBH}.$$

For LongBench, we follow the recommendation in Liu et al. (2024e) and set the maximum sequence length to 4,096 for Llama-2-7B and 8,192 for all other models. We report the KV size as the average percentage of the compressed cache relative to the FP16 cache at the end of the sequence. It includes all information needed to recover or dequantize the KV cache back to its original data types. All our experiments are conducted on a single Nvidia H100 GPU (80GB).

	$\lambda = 1 \times 10^{-4}$	$\lambda = 5 \times 10^{-4}$	$\lambda = 1 \times 10^{-3}$	$\lambda = 1 \times 10^{-2}$	$\lambda = 1 \times 10^{-1}$
$r = 5$	71.72%	73.09%	72.71%	70.96%	4.47%
$r = 10$	72.93%	71.57%	71.65%	70.81%	6.98%
$r = 20$	73.62%	72.71%	71.27%	69.22%	5.61%
$r = 40$	72.63%	72.48%	72.18%	66.95%	6.67%
$r = 60$	73.24%	72.56%	71.87%	68.68%	16.83%

Table 3: Ablation study on the GSM8k dataset using the Llama-3.1-8B-Instruct model. We vary the regularization weight λ and the subspace dimension r in SQuat.

D.3 Ablation Study

We conduct an ablation study on two most important hyper-parameters in our algorithm: the regularization weight λ in (2), and the subspace dimension r of the query matrix $\hat{\mathbf{Q}}$. Experiments are performed on the GSM8k dataset using the Llama-3.1-8B-Instruct model.

Recall that the hyper-parameter $\lambda \in [0, \infty)$ controls the trade-off between preserving the original key vectors during (de)quantization and enforcing orthogonality of the residuals to the query subspace. When $\lambda = 0$, the algorithm simplifies to a compression-based quantization method. The query subspace $\hat{\mathbf{Q}} \in \mathbb{R}^{r \times d}$ has top r important query directions. Results are summarized in Table 3. We find that performance degrades when $\lambda > 10^{-3}$. This is because $\hat{\mathbf{Q}} = \text{diag}(\boldsymbol{\Sigma}[:, r])\mathbf{V}[:, r]$ (Line 6 in Algorithm 1) and the leading singular values $\boldsymbol{\Sigma}[:, r]$ are typically large. Empirically, we observe that $\lambda = 0.001$ and $r = 5$ consistently yield strong performance. Although this configuration does not achieve the best score on GSM8K, we adopt it as the default for both SQuat and SQuat^{pre} across all models and all tasks in LM-Eval.