SVDq: 1.25-bit and 410× Key Cache Compression for LLM Attention Computation

Anonymous ACL submission

Abstract

For the efficient inference of Large Language Models (LLMs), the effective compression of key-value (KV) cache is essential. Three main types of KV cache compression techniques, namely sparsity, channel compression, and quantization, have been identified. This study presents SVDq, a Singular Value Decomposition (SVD) - based mixed precision quantization method for K cache. Initially, K cache is transformed into "latent channels" using SVD basis representations. Since the values in latent channels decay rapidly and become negligible after only a few latent channels, our method then incorporates importance-aware quantization and compression for latent channels. This enables the effective allocation of higher precision to more significant channels. Theoretically, we prove that SVDq results in quantization errors ($\times 0.1$ or even lower) that are much lower than those of per-channel key quantization in the original space. Our findings demonstrate that SVDq can achieve an equivalent key cache precision **as low as 1.25-bit**. When combined with key sparsity, it can reach a key compression ratio of up to **410× for attention computation**, all while maintaining comparable model performance. This indicates that SVDq enables high-precision low-bit quantization, providing a more efficient solution for KV cache compression in LLMs.

1 Introduction

Large Language Models (LLMs) have started a new era of artificial intelligence by demonstrating remarkable capabilities in handling complex tasks [1, 2, 3, 4]. Most of these recently developed LLMs are founded upon the attention mechanism based auto-regressive decoder transformers [5]. Consequently, they need to encode past information into intermediate hidden tensors, specifically KV caches, for subsequent and efficient inference.

However, in natural language tasks with large batches or long contexts, KV cache often expands significantly in size, posing a significant challenge to fast inference [6, 7]. The substantial memory consumption and latency required to save and load KV cache, coupled with the computational demands of attention operations, become critical bottlenecks for LLM inference. Considering the rapid advancement of computability and the increasing demand for efficient LLM inference, we recognize the importance of high-ratio KV cache compression (even with a slight concession in computational overhead), enabling the inference of LLMs on devices with limited memory.

Existing approaches to KV cache compression can be categorized into three main directions: sequence-axis compression, channel-axis compression, and digit-type compression. *(i)* Sequence-axis compression, exemplified by works such as [8, 9, 10, 11, 12, 13, 14, 15], often referred to as sparsity, involves identifying and discarding unimportant tokens for attention computation. *(ii)* Channel-axis compression, as demonstrated in, *e.g.*, [16, 17, 18], focuses on the channel dimension compression of KV cache with methods like truncating and low-rank decomposition. Notably, low-rank approximation techniques, as explored in [19, 20], represent a similar approach of this category. These methods transform KV cache into "*latent channels*" representation based on SVD, and then discard insignificant latent channels. *(iii)* Digit-type compression, also known as quantization, aims to reduce the memory footprint by employing lower-precision representations for KV cache [7, 21, 22, 23, 24]. This typically involves replacing the 32- or 16-bit FP numbers with lower precision representations. These three compression methods are proposed independently, exploiting different properties of KV cache within LLMs.

The effectiveness of quantization highly depends on the statistical distribution of the cache values. Large value ranges and outliers can lead to substantial quantization errors. In addition, the per-

formance of models degrades significantly below a certain quantization bit width (typically around 4 to 2 bits), thus limiting the compression ratio. Similarly, channel compression methods also face challenges in terms of the trade-off between accuracy and compression ratio. While works like [19, 20] have demonstrated $2\times$ compression ratios using SVD-based methods, further compression beyond this point leads to high accuracy loss. Recognizing these limitations, we emphasize the importance of combining these different strategies to further improve the compression ratio. For examples, ThinK [16] highlights the compatibility of its channel truncation method with sparsity techniques; ShadowKV [25] combines sparsity with SVD low-rank approximation to achieve minor performance degradation while achieving very high compression ratios.

In this work, we follow the channel-axis compression and quantization strategy. We find that direct truncation of the original channels, as exemplified by ThinK [16], leads to significant performance degradation when pursuing high compression ratios. To address this challenge, we propose a compression method, SVDq, that integrates the channel truncation and quantization, by utilizing our observed underlying relationship between quantization and SVD-based channel compression.

Specifically, we observe an implication of the Eckart–Young–Mirsky theorem [26]: the variances of the values within latent channels obtained through SVD are determined by the corresponding singular values and typically exhibit rapid decay. Recognizing that variances are often proportional to value ranges of latent channels, we can utilize singular values to guide the selection of quantization bit widths to balance accuracy and compression ratios.

Based on this observation, we propose a novel mixed-precision **key cache**[1] quantization method that integrates SVD-based channel compression. This method prioritizes higher bit widths for latent channels associated with larger singular values and progressively decreases precision for channels with smaller singular values. The SVD latent channels offer a significant advantage over simple variance-based descending sorting in the original space, because singular values decay exponentially for most key cache. In consequence, the range at each channel decreases fast, and often becomes insignificant after only a small number of latent channels. Hence, this approach enhances the effectiveness of quantization precision allocation for each latent channel. Furthermore, we emphasize the seamless compatibility of this method with sparsity techniques.

Our key contributions are as follows:

(1) Proposing a novel method that effectively combines quantization and latent channel compression for K cache, providing the theoretical insights.

(2) Demonstrating the compatibility of this method with sparsity techniques.

(3) Achieving a remarkable level of K cache compression with an equivalent mixed quantization precision as low as 1.25 bit while maintaining comparable model performance.

2 Related Works

Sparsity: With different feature extraction based attention estimation algorithms, methods such as Fastgen [10], H2O [9], Quest [12], SparQ [13], PQ-Cache [27], ShadowKV [25], and AttentionPredictor [15] selectively retain only the most important tokens in the sequence and effectively prune the others. Loki [14] is another sparsity method that uses the SVD approximation to accelerate attention estimation for critical tokens selection.

Channel Compression: These methods, such as ThinK [16], reduce the dimensionality of KV cache by truncating channels or employing low-rank approximations. Prominent examples include SVD-based approaches like SVD-LLM [19], LoRC [20], Palu [28], and Eigen Attention [29]. Notably, techniques like Grouped Query Attention (GQA) [30], Multi-head Latent Attention (MLA) [4], and transformations from Multi-Head Attention to GQA [31, 32] can also be viewed as forms of channel compression, as they effectively reduce the number of attention dimensions.

Quantization: Methods like KIVI [7], KVQuant [21], AlignedKV [33], BitStack [34], and KVTuner [24] reduce the memory footprint with low precision KV cache. QServe [35] introduces several quantization and system co-design methods to achieve efficient W4A8KV4, where SmoothAttention is utilized to migrate the key quantization difficulty to query.

Some works explore the combination of these approaches. In addition to the mentioned ShadowKV [25] and ThinK [16], [23] integrates quantization with matrix decomposition to apply different quan-

[1]We do not investigate the V cache since it often exhibits weak low-rank property.

tization precision for the two decomposed matrices, and Palu [28] applies per token quantization to the latent vector of the SVD low-rank approximation.

Importantly, the concept of using SVD for mixed-precision quantization has been explored in other contexts. For instance, Delta-CoMe [36] applies this principle to compress LLM weights, while SVDQuant [37] utilizes it for compressing diffusion models. The novelty of this work over the mentioned works lies not only in the application of this principle to K cache compression but also in the **theoretical foundation** upon which we derive the principle and method, and the **error analysis** we provide.

3 SVD and Quantization

Singular Value Decomposition: Let $\mathbf{K} \in \mathbb{R}^{s\times d}$ denotes the K cache matrix for a given head in a transformer layer, where s and d represent the sequence length and hidden embedding (channel) dimension, respectively, with $s \gg d$ typically holding for long context applications. Let $\mathbf{K}$ be centered by subtracting its per-channel mean $\bar{\mathbf{K}} \in \mathbb{R}^d$, *i.e.*, $\mathbf{K} \leftarrow \mathbf{K} - \bar{\mathbf{K}}$ and maintain the same notation.

Assuming $\mathbf{K}$ is full-rank. Its SVD is given by

$$\mathbf{K} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{V}^{\mathrm{H}}, \tag{1}$$

where $\mathbf{U} \in \mathbb{R}^{s\times d}$ has orthonormal columns, $\mathbf{V} \in \mathbb{R}^{d\times d}$ is orthonormal, satisfying $\mathbf{U}^{\mathrm{H}} \cdot \mathbf{U} = \mathbf{I}_d$ and $\mathbf{V}^{\mathrm{H}} \cdot \mathbf{V} = \mathbf{I}_d$, and $\mathbf{D} \in \mathbb{R}^{d\times d}$ is a diagonal matrix containing the singular values in its diagonal with elements arranged in descending order, given by $\mathbf{D} = \mathrm{Diag}([\lambda_1, ..., \lambda_d])$.

Quantization Let $\boldsymbol{k}_{\min} := (\min \mathbf{K}_{:1}, ..., \min \mathbf{K}_{:d})$, *i.e.*, the column-wise minimum vector, and analogously define $\boldsymbol{k}_{\max}$. The per-channel asymmetrical b-bit quantization and dequantization operations are given by:

$$\mathcal{Q}_b(\mathbf{K}) := \left\lfloor \frac{\mathbf{K} - \boldsymbol{k}_{\min}}{(\boldsymbol{k}_{\max} - \boldsymbol{k}_{\min})/(2^b - 1)} \right\rceil, \tag{2}$$

$$\mathcal{D}_b(\mathbf{K}_b) := \mathcal{Q}_b(\mathbf{K}) \times \frac{\boldsymbol{k}_{\max} - \boldsymbol{k}_{\min}}{2^b - 1} + \boldsymbol{k}_{\min}, \tag{3}$$

where $\lfloor \cdot \rceil$ denote the rounding operator. Naturally, $\mathcal{D}_b \circ \mathcal{Q}_b(\mathbf{K}) \approx \mathbf{K}$.

For uniformly or normally distributed columns of $\mathbf{K}$, the relative quantization errors depend solely on the bit width b, independent of the range $\boldsymbol{k}_{\max} - \boldsymbol{k}_{\min}$. However, the absolute errors scale with $\boldsymbol{k}_{\max} - \boldsymbol{k}_{\min}$, implying that smaller value ranges or variances yield smaller absolute quantization errors.

4 Methods

Although the theory of the proposed SVD-quantization method, discussed in the previous section, is expected to be applicable to a much wider range of applications, this work focuses on KV cache compression in the long context inference scenario. For long context LLMs, KV cache generated in the prefilling stage generally dominates the memory usage. Our method is proposed to address this challenge.

4.1 SVD Quantization

Consider the rows of $\mathbf{V}^{\mathrm{H}}$ in Equation (1) as a basis for the row space of $\mathbf{K}$. For the projection $\mathcal{P}_{\mathbf{V}_{:j}}$ of the rows of $\mathbf{K}$ into the j-th basis vector, defined by $\mathcal{P}_{\mathbf{V}_{:j}}(\mathbf{K}) := \mathbf{K} \cdot \mathbf{V}_{:j}$, following the Eckart–Young–Mirsky theorem [26], we have:

Theorem 4.1. *For the K cache matrix $\mathbf{K}$, the variance of its projection satisfies*

$$\mathrm{Var}(\mathcal{P}_{\mathbf{V}_{:j}}(\mathbf{K})) = \lambda_j^2. \tag{4}$$

Corollary 4.1.1. *Let $\boldsymbol{k} \in \mathbb{R}^d$ be a K cache vector with $\bar{\mathbf{K}}$ subtracted,* i.e.*, $\boldsymbol{k} \leftarrow \boldsymbol{k} - \bar{\mathbf{K}}$. For any indices $0 < i \le j < d$, the squared expectations of its projections satisfy:*

$$\mathbb{E}((\mathcal{P}_{\mathbf{V}_{:i}}(\boldsymbol{k}))^2) \ge \mathbb{E}((\mathcal{P}_{\mathbf{V}_{:j}}(\boldsymbol{k}))^2). \tag{5}$$

Proof. For any $0 < j \le d$, the projection of $\mathbf{K}$ is given by

$$\begin{aligned} \mathcal{P}_{\mathbf{V}_{:j}}(\mathbf{K}) &= \mathbf{K} \cdot \mathbf{V}_{:j} \\ &= \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{V}^{\mathrm{H}} \cdot \mathbf{V}_{:j} \\ &= \lambda_j \mathbf{U}_{:j}. \end{aligned}$$

Since $\mathbb{E}(\mathcal{P}_{\mathbf{V}_{:j}}(\mathbf{K})) = \mathcal{P}_{\mathbf{V}_{:j}}(\mathbb{E}(\mathbf{K})) = 0$, we have

$$\begin{aligned} \mathrm{Var}(\mathcal{P}_{\mathbf{V}_{:j}}(\mathbf{K})) &= \mathrm{Var}(\lambda_j \mathbf{U}_{:j}) \\ &= \lambda_j^2 \mathbb{E}\left(\mathbf{U}_{j:}^{\mathrm{H}} \cdot \mathbf{U}_{:j}\right) \\ &= \lambda_j^2. \end{aligned}$$

This proves Theorem 4.1. Corollary 4.1.1 follows directly from Theorem 4.1 when the given vector $\boldsymbol{k}$ follows the distribution of the rows of $\mathbf{K}$. □

Note that the $\mathcal{P}_{\mathbf{V}}(\mathbf{K})$ is essentially an alternative representation of $\mathbf{K}$ using the singular vector in $\mathbf{V}$ as the space bases. We call the columns of $\mathcal{P}_{\mathbf{V}}(\mathbf{K})$ *latent channels*. Figure 1 illustrates the distribution of $\mathbf{K}$ in its original space, while Figure 2

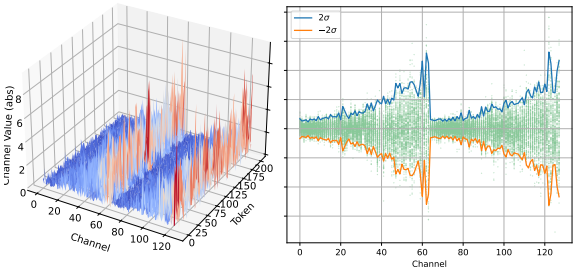

Figure 1: Original $\mathbf{K}$

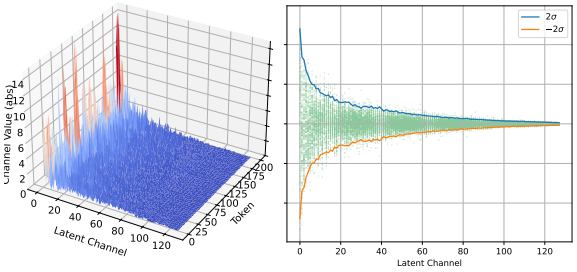

Figure 2: Projected $\mathbf{K}$, *i.e.*, $\mathcal{P}_{\mathbf{V}}(\mathbf{K})$

Figure 3: Distribution of $\mathbf{K}$ and its standard deviation

displays its representation in the SVD space after projection, demonstrating the results of Theorem 4.1 and Corollary 4.1.1. The singular vector-based projection offers a significant advantage over simple variance-based descending sorting: for most matrices, singular values typically exhibit exponential decay. Consequently, the range of projection values (represented on the y-axis in Figure 2) decreases rapidly, becoming relatively insignificant (compared to the value range of the first dimension) after only a small number of latent channels.

Since $\mathbf{K} = \mathcal{P}_{\mathbf{V}}(\mathbf{K}) \cdot \mathbf{V}^{\mathrm{H}}$ where $\mathcal{P}_{\mathbf{V}}(\mathbf{K}) := \mathbf{K} \cdot \mathbf{V}$, and all basis vectors in $\mathbf{V}$ are unit-normalized, the absolute error in approximating to $\mathcal{P}_{\mathbf{V}}$ represents both absolute and relative errors in approximating $\mathbf{K}$. Theorem 4.1, Corollary 4.1.1, and Figure 2 demonstrate that both value range and variance decay rapidly along the latent channels. This property motivates our efficient mixed-precision quantization method, SVDq, to approximate $\mathbf{K}$ via $\mathcal{P}_{\mathbf{V}}(\mathbf{K})$:

(1) Use high precision quantization for initial latent channels;

(2) Progressively decrease the precision for subsequent latent channels;

(3) Truncate the remaining latent channels with negligible value ranges or singular values.

4.2 Algorithm

In our SVDq method, we first apply SVD to the prefilling K cache, obtaining the projection operator $\mathcal{P}_{\mathbf{V}}(\cdot)$ using the right SVD matrix $\mathbf{V}$. Next, we determine a precision schedule for the quantization on each latent channel based on the singular values $[\lambda_1, ..., \lambda_d]$. Specifically, a latent channel associated with a large singular value λ is assigned a high quantization bit width b, and channels with small λ are assigned low b or even be truncated with notation $b = 0$. This yields a schedule vector $\boldsymbol{b}$, and the equivalent mixed bit width of this quantization schedule for the $\mathbf{K}$ cache is given by

$$\bar{b} = \frac{1}{d} \sum_{i=1}^{d} \boldsymbol{b}_i. \tag{6}$$

Sequently, we use $\mathcal{Q}_{\boldsymbol{b}}$ in (2) to quantize $\mathcal{P}_{\mathbf{V}}(\mathbf{K})$. The low-bit quantized $\mathcal{P}_{\mathbf{V}}(\mathbf{K})$ is then saved as the cache. In the decoding process, we dequantize the cache, reconstruct $\mathbf{K}$ in its original representation using $\mathbf{K} = \mathcal{P}_{\mathbf{V}}(\mathbf{K}) \cdot \mathbf{V}^{\mathrm{H}}$, and then proceed with the attention computation. We summarize the algorithm using pseudo-code in Algorithm 1 and an abstracted diagram in Figure 4.

Algorithm 1 SVD-quantization algorithm for $\mathbf{K}$

Require: K cache matrix $\mathbf{K}$ of L layers
Ensure: $\hat{\mathbf{K}} \approx \mathbf{K}$
  for $l \leftarrow 1$ to N **do**
    Load $\mathbf{K}$ cache matrix for l-th layer
    $\mathbf{K} = \mathbf{K} - \bar{\mathbf{K}}$
    $\mathbf{U}, \mathbf{D}, \mathbf{V} \leftarrow \mathrm{SVD}(\mathbf{K})$
    $\mathcal{P}_{\mathbf{V}}(\mathbf{K}) = \mathbf{U} \cdot \mathbf{D}$
    Set quant schedule $\boldsymbol{b}$
    Save $\bar{\mathbf{K}}$, $\mathbf{V}$, $\mathcal{Q}_{\boldsymbol{b}} \circ \mathcal{P}_{\mathbf{V}}(\mathbf{K})$, function $\mathcal{D}_{\boldsymbol{b}}$
  end for
  $\hat{\mathbf{K}} = \mathcal{D}_{\boldsymbol{b}}\left(\mathcal{Q}_{\boldsymbol{b}} \circ \mathcal{P}_{\mathbf{V}}(\mathbf{K})\right) \cdot \mathbf{V}^{\mathrm{H}} + \bar{\mathbf{K}}$

In this algorithm, the quantities to be saved include the quantized $\mathcal{P}_{\mathbf{V}}(\mathbf{K}) \in \mathbb{R}^{s \times d}$ (represented using $\bar{b}$-bit), the right SVD matrix $\mathbf{V} \in \mathbb{R}^{d \times d}$, the average of $\mathbf{K}$ denoted by $\bar{\mathbf{K}} \in \mathbb{R}^{d}$, and the dequant function $\mathcal{D}_{\boldsymbol{b}}$, which relies on the bit schedule $\boldsymbol{b} \in \mathbb{R}^{d}$ and the range of $\mathcal{P}_{\mathbf{V}}(\mathbf{K})$, given by $\boldsymbol{p}_{\min}, \boldsymbol{p}_{\max} \in \mathbb{R}^{d}$. In long context applications, $d \ll s$, the requirement of memory space for terms that depend solely on d, *e.g.*, the space for $\mathbf{V}$ and $\bar{\mathbf{K}}$, is negligible. Hence, the compression rate compared with the original 16-bit $\mathbf{K} \in \mathbb{R}^{s \times d}$ is approximately $16/\bar{b}$.

In this work, we concatenate the $\mathbf{K}$ matrices of all heads within the same layer, resulting in a larger $\mathbf{K}$ matrix with the embedding dimension d being the sum of the embedding dimensions of all heads.

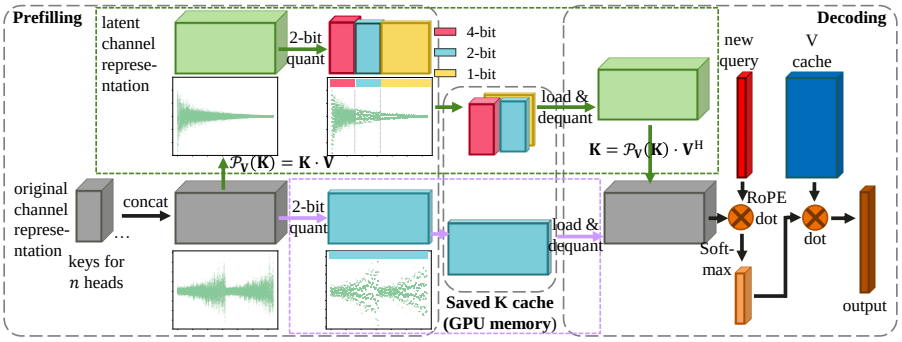

Figure 4: Diagram of SVDq method (path inside the box in green) versus direct per-channel quantization (path inside the box in violet).

To improve efficiency, for the bit schedule setting $\boldsymbol{b}$, we divide the d latent channels of $\mathcal{P}_\mathbf{V}(\mathbf{K})$ into 8 equal-sized group, each comprising $\frac{d}{8}$ dimensions. The channels within each group share the same quantization bit width. Thus, $\boldsymbol{b}$ is determined by an 8-dimensional vector $(b_1, b_2, ..., b_8)$ of integer. For example, a schedule of $(8, 4, 2, 1, 1, 0, 0, 0)$ has an equivalent mixed bit width $\bar{b} = 2$ and hence a compression ratio 8. For a model with $d = 1024$, this schedule implies:
8-bit quantization for the first 128 latent channels,
4-bit for the next 128 channels,
2-bit for the next 128 channels,
1-bit for the next 256 channels,
truncation for the remaining 384 channels.

4.3 Theoretical Error Analysis

We begin by presenting a lemma for later analysis.

Lemma 4.1. *If data X are distributed uniformly within their value range r, then the expectation of the square absolute error, ε, of an asymmetrical b-bit quantization applied to X is equal to the variance of a uniform distribution with a range of $\frac{r}{2^b}$, that is*

$$\mathbb{E}(\varepsilon^2) = \frac{1}{12}\frac{r^2}{2^{2b}}.$$

Let $\mathbf{K}$ be centered by subtracting the key's per-channel mean $\bar{\mathbf{K}} \in \mathbb{R}^d$, and let $\mathcal{P}_\mathbf{V}(\mathbf{K})$ be its latent channel representation. The Frobenius norm is invariant under this transformation, as

$$\begin{aligned}\|\mathcal{P}_\mathbf{V}(\mathbf{K})\|_\mathrm{F}^2 &= \sum_{i=1}^{s} \mathcal{P}_\mathbf{V}(\mathbf{K}_{i:}) \cdot \mathcal{P}_\mathbf{V}(\mathbf{K}_{i:})^\mathrm{H} \\ &= \sum_{i=1}^{s} \mathbf{K}_{i:} \cdot \mathbf{K}_{i:}^\mathrm{H} = \|\mathbf{K}\|_\mathrm{F}^2.\end{aligned}$$

Let $[\sigma_1^2, ..., \sigma_d^2]$ and $[\lambda_1^2, ..., \lambda_d^2]$ denote the variance vectors of the channels for the original and latent channel representations of $\mathbf{K}$, respectively. Thus,

$$\sum_{j=1}^{d} \sigma_j^2 = \frac{1}{s}\|\mathbf{K}\|_\mathrm{F}^2 = \frac{1}{s}\|\mathcal{P}_\mathbf{V}(\mathbf{K})\|_\mathrm{F}^2 = \sum_{j=1}^{d} \lambda_j^2. \quad (7)$$

We further assume that the key cache distributions in each original channel and latent channel follow uniform distributions. Then, according to Lemma 4.1, the value ranges of the j-th original channel and j-th latent channel are $r_j = 2\sqrt{3}\sigma_j$ and $\hat{r}_j = 2\sqrt{3}\lambda_j$, respectively.

Error analysis for direct quantization Figure 1 shows that the variances in the original channels often exhibit similar orders of magnitude. We therefore assume that they are approximately identical, with $\sigma_j^2 = \frac{1}{ds}\|\mathbf{K}\|_\mathrm{F}^2$ and $r_j^2 = \frac{12}{ds}\|\mathbf{K}\|_\mathrm{F}^2$. Applying a per-channel, direct b-bit quantization to $\mathbf{K}$, and following Lemma 4.1 and the above analysis, results in a quantization error ε_b with the expected value:

$$\mathbb{E}(\varepsilon_b^2) = \frac{1}{12}\frac{1}{2^{2b}}\frac{12}{ds}\|\mathbf{K}\|_\mathrm{F}^2 = \frac{1}{2^{2b}}\frac{\|\mathbf{K}\|_\mathrm{F}^2}{ds}. \quad (8)$$

Error analysis for SVDq The singular values of a matrix often exhibit exponential decay. We model the variance vector for $\mathbf{K}$'s latent channel representation as

$$\lambda_j = ce^{-\rho j} = \lambda_i e^{-\rho(j-i)}, \quad (9)$$

for any $1 \le i < j \le d$, where $c > 0$ and $\rho > 0$ are parameters.

Using this model and (7), we immediately obtain

$$c^2 = \frac{e^{2\rho} - 1}{1 - e^{-2\rho d}}\frac{\|\mathbf{K}\|_\mathrm{F}^2}{s} \approx \frac{e^{2\rho} - 1}{s}\|\mathbf{K}\|_\mathrm{F}^2,$$

as well as the square of the value range of each latent channel

$$\begin{aligned}\hat{r}_j^2 &= 12\frac{(e^{2\rho} - 1)e^{-2\rho j}}{s}\|\mathbf{K}\|_\mathrm{F}^2 \\ &= 12(e^{2\rho} - 1)e^{-2\rho j}2^{2b}d\,\mathbb{E}(\varepsilon_b^2).\end{aligned} \quad (10)$$

For further analysis, we set the bit schedule as a simple decreased arithmetic progression[2]: $b_i = (8-i)\frac{2b}{7}$, resulting in $\bar{b} = \sum_{i=1}^{8} b_i = b$, and compare SVDq with this schedule to a direct b-bit quantization. Using Lemma 4.1, for the i-th part with $\frac{d}{8}$ latent channels with quantization bit width of b_i, the expectation of the square quantization error, $\hat{\varepsilon}_i$, is

$$\begin{aligned}\mathbb{E}(\hat{\varepsilon}_i^2) &= \frac{8}{d}\sum_{j=d(i-1)/8+1}^{di/8}\frac{1}{12}\frac{\hat{r}_j^2}{2^{2b_i}} \\ &= 8\frac{e^{2\rho}-1}{2^{2(b_i-b)}}\mathbb{E}(\varepsilon_b^2)\sum_{j=d(i-1)/8+1}^{di/8} e^{-2\rho j} \\ &\approx 8\frac{e^{-d\rho(i-1)/4}}{e^{(b_i-b)\ln 4}}\mathbb{E}(\varepsilon_b^2).\end{aligned}$$

Denoting $\hat{b}_i := b_1 - b_i = (i-1)\frac{2b}{7}$ and $\alpha := \frac{d\rho}{4} - \frac{2b}{7}\ln 4$, the error for SVDq, $\hat{\varepsilon}_b$, satisfies

$$\begin{aligned}\mathbb{E}(\hat{\varepsilon}_b^2) &= \frac{1}{8}\sum_{i=1}^{8}\mathbb{E}(\hat{\varepsilon}_i^2) \\ &= \mathbb{E}(\varepsilon_b^2)\sum_{i=1}^{8}\frac{e^{-d\rho(i-1)/4}}{e^{(b_i-b)\ln 4}} \\ &= \frac{\mathbb{E}(\varepsilon_b^2)}{4^{b_1-b}}\sum_{i=1}^{8} e^{-d\rho(i-1)/4+\hat{b}_i\ln 4} \\ &= \frac{\mathbb{E}(\varepsilon_b^2)}{4^{b_1-b}}\sum_{i=1}^{8} e^{-\alpha(i-1)} \\ &= \frac{1}{4^{b_1-b}}\frac{1-e^{-8\alpha}}{1-e^{-\alpha}}\mathbb{E}(\varepsilon_b^2).\end{aligned}$$

For LLMs like Llama-3.1-8B, $d = 1024$, the decay rate ρ is often on the order of approximately 0.1, while we typically consider quantization bit widths at the levels $b = 2$ or 4. Consequently, we often have $\rho \gg \frac{8b}{7d}\ln 4$, resulting in $\alpha \gg 0$. Under these conditions, typically $\left(\frac{\mathbb{E}(\hat{\varepsilon}_b^2)}{\mathbb{E}(\varepsilon_b^2)}\right)^{\frac{1}{2}} \approx 2^{b-b_1} < 0.1$, the expectation quantization error of SVDq is much smaller than the direct per-channel quantization error. This result theoretically proves the efficiency of mixed-precision quantization in the latent channel representation guided by SVD.

[2]This setting is introduced only for the sake of clear theoretical error analysis, as it yields concise error expressions. It is not a realistic schedule because it may contain no integer bit widths. A similar analysis can be applied to other schedules, although the derivations may become more complex.

Model	d_h	n	d	part dim $\frac{d}{8}$
Llama-3.1-8B-Instruct	128	8	1024	128
Qwen2.5-7B-Instruct	128	4	512	64
Qwen2.5-3B-Instruct	128	2	256	32

Table 1: Configuration of K cache for three models.

5 Experiments

In this section, we apply our method in different model settings to showcase its efficiency in K cache compression.

We focus on long context applications using three large language models: Llama-3.1-8B-Instruct [2], Qwen2.5-7B-Instruct, and Qwen2.5-3B-Instruct [3]. The numerical experiments are based on the RULER benchmarks [38] and LongBench benchmarks [39]. We omit the scores for RULER NIAH Single tests because in our tests, almost all methods achieved perfect scores (100) on these tests, indicating that they do not pose a sufficient challenge. We present the results of RULER in this section and refer the readers to Appendix B for the results of LongBench.

The configuration settings for the K cache of the three models are listed in Table 1. The long context prompt length is set to 64K, satisfying $s = 64 \times 1024 \gg d$.

5.1 Results of SVDq

In our first experiment, we implement the SVD quantization method directly in K cache compression and summarize the results in Table 2. Detailed experiment settings and descriptions are provided in the Appendix A.1.

The results demonstrate that the proposed SVDq method generally results in lower performance degradation compared to direct quantization and channel compression across almost all tests. On average, the SVDq method achieves higher scores despite having a lower equivalent mixed quantization bit width. This clearly showcases the significant advantage of truncating and quantizing the SVD latent channels over operating directly on the original channels.

Please note that in our tests, both direct 2-bit quantization of the original **K** and equivalent 2-bit ThinK that retains $\frac{1}{2}$ original channels and combines 4-bit quantization result in much more significant performance degradation. Therefore, we opted to compare our SVDq method in 2- and 3-bit setting with direct 3-bit quantization and equivalent 3-bit ThinK for a more meaningful evaluation.

Method	bit	CR	N-MK1	N-MK2	N-MQ	N-MV	VT	FWE	QA-1	QA-2	Average
Llama-3.1-8B-Instruct											
Default	16	1.0	99.0	97.9	98.7	98.2	97.5	85.4	82.3	60.4	90.0
Per-channel Quant	3	5.3	97.9	70.8	94.0	91.1	86.0	84.7	67.7	46.9	79.9
ThinK	3	5.3	94.8	66.7	87.5	80.7	66.2	90.3	75.0	55.2	77.2
SVDq (ours)	3	5.3	100.0	96.9	99.2	95.3	97.3	86.1	85.4	57.3	89.7
SVDq (ours)	2	8.0	99.0	94.8	96.1	92.7	99.0	84.4	75.0	47.9	86.1
Qwen2.5-7B-Instruct											
Default	16	1.0	86.5	26.0	95.8	87.5	85.8	83.0	61.5	38.5	70.6
Per-channel Quant	3	5.3	37.5	3.1	46.9	47.7	63.5	77.1	18.8	25.0	39.9
ThinK	3	5.3	60.4	8.3	66.9	71.1	63.7	76.7	40.6	35.4	52.9
SVDq (ours)	3	5.3	88.5	29.2	92.7	80.2	84.0	87.8	54.2	40.6	69.7
SVDq (ours)	2	8.0	78.1	36.5	81.8	82.6	79.4	71.5	39.6	32.3	62.7
Qwen2.5-3B-Instruct											
Default	16	1.0	78.1	27.1	89.8	88.8	81.0	72.2	41.7	30.2	63.6
Per-channel Quant	3	5.3	27.1	3.1	23.2	25.8	61.7	63.2	14.6	24.0	30.3
ThinK	3	5.3	38.5	7.3	49.5	47.9	64.8	66.3	26.0	25.0	40.7
SVDq (ours)	3	5.3	66.7	15.6	79.7	75.3	74.2	66.7	24.0	27.1	53.6
SVDq (ours)	2	8.0	52.1	16.7	57.8	56.0	69.8	58.7	19.8	27.1	44.7

Table 2: Performance of our method ("SVDq") for key compression in different models on the RULER benchmark evaluated at a context length of 64K. The bit schedules for SVDq are $\boldsymbol{b} = (8, 4, 4, 4, 2, 2, 0, 0), (8, 4, 4, 0, 0, 0, 0, 0)$, resulting in $\bar{b} = 3, 2$, respectively. The third column ("CR") is refer to as compression ratio given by $16/\bar{b}$. The second row ("Per-channel Quant") refers to applying direct per-channel quantization to the original **K**. The thrid row ("ThinK") refers to applying ThinK method [16] with $\frac{3}{4}$ compression ratio to the original **K**, combining 4-bit quantization. Our method outperforms direct quantization and ThinK with quantization despite having a lower (mixed) bit width (2 bits versus 3 bits). The value cache is retained in BF16 type without any processes. Detailed settings are found in the Appendix A.1.

5.2 Results of SVDq with Sparsity

Although SVDq can improve model performance while using small bit quantizations, significant performance loss can still occur when the bit width is extremely low, such as $\bar{b} = 2$. Hence, we combine our SVDq method with a sparsity technique to investigate its compatibility with other techniques and explore potential performance improvements.

We adopt the sparsity strategy proposed in the ShadowKV method [25]. Table 3 presents the results for sparsity and ShadowKV as baselines. Please see a brief introduction of the ShadowKV and the description of these baseline settings in the Appendix A.2. For the SVDq method, we investigate different quantization bit schedules with varying equivalent mixed bit widths: $\bar{b} = 2.25, 1.75$, and 1.25. Detailed schedules are provided in Table 4 in the Appendix A.2. We apply the SVDq in conjunction with the sparsity method from ShadowKV. The scores are also presented in Table 3.

Our observations reveal that, when combined with sparsity, our SVDq compression method does not result in significant performance degradation, even with extremely low quantization bit widths such as $\bar{b} = 1.25$. Decreasing the bit width from $\bar{b} = 2.25$ to $\bar{b} = 1.75$ has a negligible impact on the score. Further decreasing $\bar{b}$ to 1.25 results in a slight performance loss, although it remains relatively insignificant. Notably, our quantization method, even with $\bar{b} = 1.25$, outperforms the low-rank approximation used in ShadowKV, demonstrating the ineffectiveness of directly truncating SVD ranks. Taking into account the sparsity compression ratio of $32\times$, SVDq contributes an additional ratio of up to $12.8\times$, resulting in a total compression ratio of $400\times$.

Notably, by comparing Tables 2 and 3, the introduction of sparsity does not result in performance degradation; it can even improve the performance of models that solely use SVDq or low-rank compression. We observe that with sparsity, the model can withstand higher compression ratios. This may be attributed to the fact that quantization and low-rank approximation introduce errors across all tokens, potentially leading to significant error accumulation in the full attention mechanism. However, sparsity discards unimportant tokens, which can help to mitigate the error from these tokens and improve overall performance.

5.3 Results of SVDq with Sparsity and V Quantization

In the final experiment, we repeat the second experiment while additionally introducing a quantization method to the V cache to further reduce the required memory for model loading. Please find the experiment in Appendix A.3.

Method	bit	CR	N-MK1	N-MK2	N-MQ	N-MV	VT	FWE	QA-1	QA-2	Average
Llama-3.1-8B-Instruct											
Default	16	1	99.0	97.9	98.7	98.2	97.5	85.4	82.3	60.4	90.0
ShadowKV Sparsity	16	32	100.0	97.9	99.0	94.5	89.6	74.0	82.3	61.5	87.3
ShadowKV	2.5	205	99.0	97.9	99.0	96.1	85.6	75.0	82.3	59.4	86.8
SVDq+Sparsity	2.25	227	100.0	97.9	98.4	95.3	89.6	74.0	81.2	60.4	87.1
SVDq+Sparsity	1.75	291	100.0	97.9	98.7	94.5	88.7	74.7	83.3	60.4	87.3
SVDq+Sparsity	1.25	410	99.0	96.6	99.2	93.2	87.3	74.3	83.3	60.4	86.7
Qwen2.5-7B-Instruct											
Default	16	1	86.5	26.0	95.8	87.5	85.8	83.0	61.5	38.5	70.6
ShadowKV Sparsity	16	32	85.4	19.8	93.5	87.2	86.9	70.8	65.6	35.4	68.1
ShadowKV	2.5	205	86.5	17.7	89.8	75.8	71.2	62.8	67.7	37.5	63.6
SVDq+Sparsity	2.25	227	89.6	19.8	94.3	89.6	85.6	69.1	67.7	38.5	69.3
SVDq+Sparsity	1.75	291	87.5	15.6	94.3	88.5	81.9	69.1	65.6	37.5	67.5
SVDq+Sparsity	1.25	410	86.5	15.6	93.5	88.0	83.7	68.1	62.5	36.5	66.8
Qwen2.5-3B-Instruct											
Default	16	1	78.1	27.1	89.8	88.8	81.0	72.2	41.7	30.2	63.6
ShadowKV Sparsity	16	32	77.1	18.8	83.6	81.8	75.2	48.6	43.8	28.1	57.1
ShadowKV	2.5	205	75	17.7	69.3	71.4	69.2	50.7	32.3	29.2	51.8
SVDq+Sparsity	2.25	227	78.1	19.8	82.0	83.6	77.3	47.2	36.5	28.1	56.6
SVDq+Sparsity	1.75	291	80.2	20.8	80.7	83.3	76.9	49.7	38.5	27.1	57.2
SVDq+Sparsity	1.25	410	75.0	17.7	78.9	82.6	77.1	46.9	35.4	30.2	55.5

Table 3: Performance of our method in conjunction with the sparsity strategy of ShadowKV, denoted by "SVDq+Sparsity", in different models on the RULER benchmark evaluated at a context length of 64K. The third column key compression ratio ("CR") is computed by $16/\bar{b}\times$ the sparsity ratio, 32, and represents the compression ratio of the key cache that involves in the attention computation. The second row ("ShadowKV Sparsity") refers to applying only the sparsity strategy of ShadowKV without any quantization or SVD low-rank methods. For the third row ("ShadowKV"), in the Llama-3.1 model, we use the same settings as in the ShadowKV paper, retaining 160 ranks of the SVD and truncating the rest, which is equivalent to a quantization bit width of 2.5. For the Qwen2.5-7B and 3B models, to maintain consistent quantization bit widths (2.5 bits), we retain 80 and 40 ranks, respectively. The quantization bit schedules for "SVDq+Sparsity" are identical for all three models and are shown in Table 4 in Appendix A.2. Our method outperforms ShadowKV despite having a lower (mixed) bit width.

The resulting insignificant performance degeneration not only demonstrate the effectiveness of the SVD quantization method in K cache compression but also highlight its compatibility with existing compression techniques.

5.4 Results of LongBench benchmark

Numerical experiments based on the LongBench benchmark [39] are presented in Appendix B and Table 6. We observe results consistent with those obtained on the RULER benchmark. For most of the models and method configurations, our SVDq method either outperforms or exhibits comparable performance to the baselines, including per-channel quantization [40], ThinK [16], and ShadowKV [18]. Notably, the performance degradation of our method compared to the full, non-quantized model is insignificant and nearly lossless for LongBench datasets. These results further corroborate the conclusions drawn from our analysis of the RULER benchmark.

6 Conclusions

We present a mixed precision quantization approach for KV cache compression, which is grounded in projection representation within the SVD and singular vector space. In this method, we assign higher quantization bit widths to the initial latent channels and gradually reduce the bit widths for subsequent latent channels. Additionally, there is an option to truncate the final channels. Through comprehensive experiments, we show that this approach outperforms direct per - channel quantization in terms of model performance, even when using lower mixed bit widths. Moreover, we explore the performance of our proposed method when integrated with other KV cache compression techniques, such as sparsity and V cache quantization. Our results reveal that our method incurs minimal performance degradation, even when extremely low equivalent quantization bit widths (mixed 1.75 and 1.25 bits for the K cache) are utilized. Overall, these findings convincingly demonstrate the effectiveness and efficiency of our proposed method in K cache compression.

7 Limitations

Although our method demonstrates good effectiveness in K cache compression, it primarily reduces the required memory space for model loading without directly addressing computational cost. In fact, our current implementation may even slightly increase inference time.

Specifically, we utilize the pre-RoPE setting in our implementation. Our method extracts a quantized low-bit K cache of the SVD projection representation before the application of Rotary Position Embeddings (RoPE) and shares this low-bit representation across all heads. Due to the online computation of RoPE, which depends on the incoming position index, the reconstruction from the projection representation to the original representation cannot be efficiently integrated into the model's forward pass. Consequently, this leads to an increase in computational cost for each head.

This increase in computational cost could potentially be remedied by switching to the post-RoPE setting, where K cache is handled after the application of RoPE. However, as reported in ShadowKV work [25] and observed in our numerical tests, the post-RoPE setting generally exhibits degraded performance compared to the pre-RoPE setting.

Therefore, investigating methods to accelerate the computation of our SVD quantization method, potentially by exploring alternative approaches or optimizations within the pre-RoPE framework, is an interesting direction for future research.

References

[1] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, et al. GPT-4 Technical Report, March 2024. arXiv:2303.08774 [cs].

[2] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 Herd of Models, November 2024. arXiv:2407.21783 [cs].

[3] Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. Qwen2.5 Technical Report, January 2025. arXiv:2412.15115 [cs].

[4] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025. Version Number: 1.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. arXiv:1706.03762 [cs].

[6] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently Scaling Transformer Inference, November 2022. arXiv:2211.05102 [cs].

[7] 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, 2023. arXiv:2402.02750 [cs].

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

[9] 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, 2024.

[10] 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.

[11] 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.

[12] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-Aware Sparsity for Efficient Long-Context LLM Inference, August 2024. arXiv:2406.10774 [cs].

[13] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ Attention: Bandwidth-Efficient LLM Inference, September 2024. arXiv:2312.04985 [cs].

[14] Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank Keys for Efficient Sparse Attention, November 2024. arXiv:2406.02542 [cs].

[15] Qingyue Yang, Jie Wang, Xing Li, Zhihai Wang, Chen Chen, Lei Chen, Xianzhi Yu, Wulong Liu, Jianye Hao, Mingxuan Yuan, et al. Attentionpredictor: Temporal pattern matters for efficient llm inference. *arXiv preprint arXiv:2502.04077*, 2025.

[16] 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, October 2024. arXiv:2407.21018 [cs].

[17] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[18] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*, 2024.

[19] Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware Singular Value Decomposition for Large Language Model Compression, May 2024. arXiv:2403.07378 [cs].

[20] Rongzhi Zhang, Kuang Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and Yelong Shen. LoRC: Low-Rank Compression for LLMs KV Cache with a Progressive Compression Strategy, October 2024. arXiv:2410.03111 [cs].

[21] 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*, 2024.

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

[23] Peiyu Liu, Ze-Feng Gao, Wayne Xin Zhao, Yipeng Ma, Tao Wang, and Ji-Rong Wen. Unlocking Data-free Low-bit Quantization with Matrix Decomposition for KV Cache Compression, May 2024. arXiv:2405.12591 [cs].

[24] Xing Li, Zeyu Xing, Yiming Li, Linping Qu, Hui-Ling Zhen, Wulong Liu, Yiwu Yao, Sinno Jialin Pan, and Mingxuan Yuan. Kvtuner: Sensitivity-aware layer-wise mixed precision kv cache quantization for efficient and nearly lossless llm inference, 2025.

[25] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV Cache in Shadows for High-Throughput Long-Context LLM Inference, October 2024. arXiv:2410.21465 [cs].

[26] L. Mirsky. SYMMETRIC GAUGE FUNCTIONS AND UNITARILY INVARIANT NORMS. *The Quarterly Journal of Mathematics*, 11(1):50–59, 1960.

[27] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. Pqcache: Product quantization-based kvcache for long context llm inference. *arXiv preprint arXiv:2407.12820*, 2024.

[28] 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, November 2024. arXiv:2407.21118 [cs].

[29] Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen Attention: Attention in Low-Rank Space for KV Cache Compression, November 2024. arXiv:2408.05646 [cs].

[30] 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, December 2023. arXiv:2305.13245 [cs].

[31] Qingyun Jin, Xiaohui Song, Feng Zhou, and Zengchang Qin. Align Attention Heads Before Merging Them: An Effective Way for Converting MHA to GQA, December 2024. arXiv:2412.20677 [cs].

[32] Yuang Chen, Cheng Zhang, Xitong Gao, Robert D. Mullins, George A. Constantinides, and Yiren Zhao. Optimised Grouped-Query Attention Mechanism for Transformers, June 2024. arXiv:2406.14963 [cs].

[33] Yifan Tan, Haoze Wang, Chao Yan, and Yangdong Deng. AlignedKV: Reducing Memory Access of KV-Cache with Precision-Aligned Quantization, October 2024. arXiv:2409.16546 [cs].

[34] Xinghao Wang, Pengyu Wang, Bo Wang, Dong Zhang, Yunhua Zhou, and Xipeng Qiu. BitStack: Fine-Grained Size Control for Compressed Large Language Models in Variable Memory Environments, October 2024. arXiv:2410.23918 [cs].

[35] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. QServe: W4A8KV4 Quantization and System Co-design for Efficient LLM Serving, May 2024. arXiv:2405.04532 [cs].

[36] Bowen Ping, Shuo Wang, Hanqing Wang, Xu Han, Yuzhuang Xu, Yukun Yan, Yun Chen, Baobao Chang, Zhiyuan Liu, and Maosong Sun. Delta-CoMe: Training-Free Delta-Compression with Mixed-Precision for Large Language Models, November 2024. arXiv:2406.08903 [cs].

[37] Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. SVDQuant: Absorbing Outliers by Low-Rank Components for 4-Bit Diffusion Models, November 2024. arXiv:2411.05007 [cs].

[38] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the Real Context Size of Your Long-Context Language Models?, August 2024. arXiv:2404.06654 [cs].

[39] 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. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

[40] 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*, 2024.

A Experiments Descriptions

A.1 Descriptions for Section 5.1

In this experiment, we include the below baselines for comparison:

Default No compression is applied, and 16-bit widths are used for all values. This is the default configuration of each models;

Direct 3-bit Quantization 3-bit per-channel quantization [7] is applied directly to the **K** matrix in its original space (as depicted in Figure 1).

ThinK Direct channel truncation in the original space by ThinK [16] that retains $\frac{3}{4}$ channels, in conjunction with 4-bit quantization, results in an equivalent 3-bit setting.

The equivalent mixed quantization bit width in this experiment are selected as $\bar{b} = 3, 2$ for the SVDq method. The quantization schedule $\boldsymbol{b}$ is set to $(8, 4, 4, 4, 2, 2, 0, 0)$ and $(8, 4, 4, 0, 0, 0, 0, 0)$, respectively.

A.2 Descriptions for Section 5.2

ShadowKV [25] and its sparsity techniques act as baselines and utilized in this work. Briefly, this strategy divides the K cache in the prefilling stage into small chunks, each containing 8 tokens. It computes the mean embedding of each trunk as the landmark and then uses these landmarks to identify important chunks. Specifically, the top-k chunks with the highest attention scores are considered important and retained, while the remaining chunks are neglected in the computation of attention. Note that this method also includes an auxiliary selection mechanism for outlier chunks, which are identified based on low cosine similarity. These outliers are not clipped during the sparsity process. In addition to sparsity, the full ShadowKV method incorporates SVD low-rank approximation of the K cache, retaining 160 out of the full 1024 ranks. This low-rank approximation can be considered equivalent to approximately 2.5-bit quantization, as the default numerical precision is 16 bits.

Based on ShadowKV, the baseline results for comparison that shown in Table 3 (the first three rows of each model) are:

Default Scores obtained with the default 16-bit digital precision;

Sparsity Scores obtained using the ShadowKV sparsity method without low-rank approximation or quantization;

ShadowKV Scores obtained using the full ShadowKV method, including both sparsity and equiva-

Equivalent bit $\bar{b}$	schedule $\boldsymbol{b}$
2.25	$(8, 4, 4, 2, 0, 0, 0, 0)$
1.75	$(8, 4, 2, 0, 0, 0, 0, 0)$
1.25	$(4, 4, 2, 0, 0, 0, 0, 0)$

Table 4: Key quantization bit schedules for SVDq.

lent 2.5-bit quantization.

The detailed quantization schedules are shown in Table 4.

A.3 Descriptions and Results for Section 5.3

In this experiment, the configuration of K cache compression and sparsity remains the same as in the second experiment: the mixed quantization bit schedules are set according to Table 4, consistent with the previous experiment, and the sparsity method employs the ShadowKV sparsity technique [25]. In addition to these settings, we observe the very weak low-rank property of V cache and hence apply a direct 4-bit per-token quantization to the V cache. The results are presented in Table 5.

Our observations indicate a very small performance loss in Table 5 compared to the obtained scores in Table 3. This suggests that, despite being an approximation method with a very low compression rate, SVDq does not significantly degrade model performance even when combined with sparsity and V cache compression.

B Experiment in LongBench

We also implement numerical experiments based on the LongBench benchmark [39] and exclude the tests of which the sequence lengths are less than 4K. The baselines and configurations of our method are the same as those presented in Section 5. The results are shown in Table 6. Note that the second row for each model, which includes the results for "ShadowKV Sparsity," "ShadowKV," and three "SVDq+Sparsity" configurations, corresponds to the results in Table 3. Similarly, the third row for each model, which includes the results for three "SVDq+Sparsity" configurations, corresponds to the results in Table 5, which applies an additional 4-bit quantization for V cache comparing to the second row. For most of the models and method configurations, our SVDq method outperforms the other methods, demonstrating the same conclusions as observed in the numerical experiments on the RULER benchmark.

Method	bit	CR	N-MK1	N-MK2	N-MQ	N-MV	VT	FWE	QA-1	QA-2	Average
Llama-3.1-8B-Instruct											
Default	16	1	99.0	97.9	98.7	98.2	97.5	85.4	82.3	60.4	90.0
SVDq+Sparsity+V4	2.25	227	100.0	97.9	98.4	95.3	88.3	75.0	81.2	60.4	87.1
SVDq+Sparsity+V4	1.75	291	100.0	96.9	99.0	94.5	87.7	75.7	832.3	60.4	87.1
SVDq+Sparsity+V4	1.25	410	99.0	96.9	99.2	93.0	86.2	73.3	83.3	60.4	86.4
Qwen2.5-7B-Instruct											
Default	16	1	86.5	26.0	95.8	87.5	85.8	83.0	61.5	38.5	70.6
SVDq+Sparsity+V4	2.25	227	86.5	20.8	95.1	89.6	84.4	70.1	66.7	39.6	69.1
SVDq+Sparsity+V4	1.75	291	86.5	18.8	93.5	90.4	82.5	68.1	64.6	36.5	67.6
SVDq+Sparsity+V4	1.25	410	86.5	16.7	92.4	87.0	83.3	68.4	62.5	39.6	67.0
Qwen2.5-3B-Instruct											
Default	16	1	78.1	27.1	89.8	88.8	81.0	72.2	41.7	30.2	63.6
SVDq+Sparsity+V4	2.25	227	75.0	20.5	80.2	83.9	78.7	45.8	38.5	28.1	56.4
SVDq+Sparsity+V4	1.75	291	80.2	19.8	81.8	83.3	76.0	49.0	37.5	29.2	57.1
SVDq+Sparsity+V4	1.25	410	77.1	13.5	77.3	81.2	76.2	46.9	33.3	29.2	54.4

Table 5: Performance of the "SVDq+Sparsity" method in conjunction with a 4-bit per-token quantization method in the V cache in different models on the RULER benchmark evaluated at a context length of 64K. The quantization bit schedules for "SVDq+Sparsity+V4" are the same as those used in Table 3 and are shown in Table 4. Our method is perfectly compatible with the V cache quantization method, leading to negligible performance loss compared to the results shown in Table 3.

Method	bit	CR	NarrativeQA	HotpotQA	MuSiQue	GovRepprt	SAMSum	RepoBench-P	Average
Llama-3.1-8B-Instruct									
Default	16	1	22.3	17.5	14.2	33.4	35.7	43.4	30.3
Per-channel Quant	16	1.0	17.7	15.9	6.15	33.0	35.4	30.9	24.1
ThinK	3	5.3	14.0	15.4	11.0	33.0	35.2	48.8	30.0
SVDq	3	5.3	20.2	16.3	11.0	34.2	35.3	45.1	30.0
SVDq	2	8.0	18.4	18.0	11.5	32.3	34.7	48.5	30.8
ShadowKV Sparsity	16	32	1.91	4.08	1.99	9.05	6.09	18.3	8.90
ShadowKV	2.5	205	22.6	21.5	10.7	32.5	37.1	45.6	31.2
SVDq+Sparsity	2.25	227	22.3	21.4	9.54	33.2	36.2	42.3	29.9
SVDq+Sparsity	1.75	291	22.8	21.3	10.3	33.4	35.2	43.7	30.4
SVDq+Sparsity	1.25	410	20.8	17.9	11.1	33.0	34.2	43.1	29.4
SVDq+Sparsity+V4	2.25	227	22.0	19.6	13.1	33.6	35.4	41.3	29.7
SVDq+Sparsity+V4	1.75	291	22.3	19.5	11.4	33.6	34.9	44.1	30.3
SVDq+Sparsity+V4	1.25	410	20.9	22.1	11.9	33.1	37.0	41.8	30.0
Qwen2.5-7B-Instruct									
Default	16	1	8.78	11.2	7.35	31.5	40.1	49.3	28.7
Per-channel Quant	16	1.0	6.46	12.3	5.69	30.6	41.1	44.3	26.6
ThinK	3	5.3	5.42	8.97	5.02	29.8	30.4	35.7	21.8
SVDq	3	5.3	8.80	11.3	8.32	31.1	40.2	48.9	28.6
SVDq	2	8.0	6.84	19.9	9.47	31.9	40.4	48.5	29.7
ShadowKV Sparsity	16	32	10.5	10.5	7.78	31.8	38.9	49.9	29.0
ShadowKV	2.5	205	10.3	12.0	8.06	30.9	40.1	49.1	29.0
SVDq+Sparsity	2.25	227	11.3	11.2	7.10	31.4	41.5	50.6	29.6
SVDq+Sparsity	1.75	291	10.3	11.5	7.14	31.5	39.7	52.1	29.7
SVDq+Sparsity	1.25	410	9.74	11.0	7.74	31.5	40.7	51.5	29.6
SVDq+Sparsity+V4	2.25	227	10.5	11.2	8.49	31.6	40.5	51.4	29.8
SVDq+Sparsity+V4	1.75	291	7.83	10.5	7.83	31.3	40.1	53.5	29.8
SVDq+Sparsity+V4	1.25	410	9.59	10.8	7.37	31.0	40.7	52.5	29.8
Qwen2.5-3B-Instruct									
Default	16	1	6.87	14.4	10.1	30.6	37.6	46.1	27.8
Per-channel Quant	16	1.0	6.32	9.47	4.13	29.2	35.6	44.6	25.3
ThinK	3	5.3	6.39	8.11	5.72	29.8	36.3	43.9	25.2
SVDq	3	5.3	7.33	14.5	7.55	29.9	35.8	48.2	27.9
SVDq	2	8.0	3.26	8.06	5.17	26.1	35.3	53.0	27.0
ShadowKV Sparsity	16	32	8.32	14.2	8.54	29.8	37.7	50.0	28.9
ShadowKV	2.5	205	7.19	15.8	9.04	27.4	37.5	47.2	27.8
SVDq+Sparsity	2.25	227	7.14	15.2	9.76	30.0	38.3	46.7	28.1
SVDq+Sparsity	1.75	291	7.51	15.0	7.27	29.3	37.6	46.6	27.5
SVDq+Sparsity	1.25	410	8.15	14.9	8.09	29.3	38.4	48.2	28.4
SVDq+Sparsity+V4	2.25	227	7.22	14.0	8.45	29.0	38.2	47.3	27.8
SVDq+Sparsity+V4	1.75	291	6.41	14.3	10.3	29.2	35.4	48.1	27.9
SVDq+Sparsity+V4	1.25	410	7.68	13.9	8.26	29.1	37.4	46.8	27.6

Table 6: Results of the tests on LongBench benchmarks [39] (longer than 4K). The experiment settings are the same as those for RULER benchmarks in Section 5. The second row for each model, which includes the results for "ShadowKV Sparsity," "ShadowKV," and three "SVDq+Sparsity" configurations, corresponds to the results in Table 3. Similarly, the third row for each model, which includes the results for three "SVDq+Sparsity+V4" configurations, corresponds to the results in Table 5, which applies an additional 4-bit quantization for V cache comparing to the second row.