XQuant: Pushing Low-Bit KV Cache Quantization to the Limit with Cross-Layer Compression

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse natural language processing tasks. However, their extensive memory requirements stemming from KV cache growth, especially during long-text understanding and generation, pose significant challenges for real-world deployment in resource-constrained environments. Quantization, as a promising approach that preserves historical information while reducing memory consumption, has garnered significant attention and expectations. We present XQuant, a training-free and plug-and-play framework that pushes KV cache quantization to ultra-low equivalent bit-width. XQuant introduces two key improvements over existing quantization methods: a computationally negligible data-free calibration approach and cross-layer compression sharing, enabling ultra-low equivalent bit-width. Extensive experiments on CoQA, TruthfulQA, and LongBench demonstrate that XQuant achieves lower equivalent bit-width (< 1.4 bits) across various large language models compared to KIVI-2bit and AsymKV-1.5bit baselines, while attaining superior performance metrics, establishing a better trade-off between model performance and compression ratio.

1 Introduction

The rapid advancement of Large Language Models (LLMs) has propelled significant progress in a wide array of natural language processing (NLP) applications, including code generation, search systems, and many others (Ouyang et al., 2023; Sharma et al., 2024; Ma et al., 2024). The exceptional performance of LLMs is primarily driven by their immense parameter scales, which enable them to excel across diverse tasks. However, this remarkable success comes with substantial costs: the computational and memory demands associated with deploying LLMs have increased exponentially due to increasong models parameters and growing input and output, posing a formidable bottleneck for practical deployment. In particular, GPU memory consumption has surged to levels that frequently surpass the capacities of current hardware infrastructures, making large-scale deployment increasingly challenging (Shi et al., 2024).

To mitigate this challenge, the Key-Value (KV) cache mechanism has been widely adopted (Yao et al., 2024; Yang et al., 2024d; Ainslie et al., 2023; Kwon et al., 2023). The KV cache optimizes memory efficiency by storing and reusing previously computed keys and values in the attention mechanism, thereby reducing redundant computations and GPU memory usage. Despite its advantages, as model sizes and the input/output sequence lengths continue to grow, the storage overhead of the KV cache itself becomes increasingly significant (Shi et al., 2024). For instance, a 30-billion-parameter language model with a batch size of 128 and a sequence length of 1024 may require up to 180 GB of memory solely for storing the KV cache (Zhang et al., 2023). Although the computational and memory requirements are reduced compared to not using it, such escalating demands still pose substantial challenges for deploying LLMs with constrained hardware resources.

To address this problem, prior works have explored various strategies from different perspectives. Some studies (Sheng et al., 2023; Hooper et al., 2024; Liu et al., 2024b; Tao et al., 2024) focus on quantizing the floating-point KV cache (and, in some cases, model weights) to lower precision. However, these approaches often experience performance degradation under extreme compression ratios, particularly around 2-bit precision. Alternatively, other methods (Xiao et al., 2023; Zhang et al., 2023; Li et al., 2024; Cai et al., 2024) aim to alleviate the storage burden by evicting unimportant tokens. These methods dynamically or statically identify and discard less critical tokens to reduce memory usage. Nevertheless, these methods inherently introduce information loss, resulting

in reduced memory retention and severe forgetting issues, which can undermine the model's ability to maintain consistent performance on longer sequences. Existing approaches to KV cache quantization often prioritize computational efficiency but tend to sacrifice precision, particularly when operating under ultra-low-bit settings.

To address these limitations, this paper focuses on training-free KV cache quantization scenarios under extreme compression ratios and introduces **XQuant, a plug-and-play framework for ultra-low-bit KV cache quantization.** XQuant delivers two key improvements over existing quantization methods: **(1) Data-Free Calibration:** Traditional quantization methods often face significant limitations when mapping values to low-bit precision. Specifically, they tend to use the two endpoint values (e.g., 0 and 1 in 1-bit quantization) as representative values, which can result in substantial quantization errors, particularly under low-bit width settings. To address this issue, XQuant introduces a parameterized calibration scheme that allows for more fine-grained mapping of values. By adjusting the representative values to better reflect the actual data distribution, this method significantly reduces quantization errors and minimizes performance loss without the need for additional data. **(2) Cross-Layer KV Cache Sharing:** We observe enhanced KV cache similarity between adjacent layers after quantization - a previously overlooked phenomenon. This enables effective cross-layer compression where one layer's quantized KV cache is shared across subsequent layers, significantly reducing computational and memory costs while preserving model performance.

To evaluate the effectiveness of XQuant, we conduct extensive experiments on a consumer-grade NVIDIA GeForce RTX 3090 GPU (24GB) across diverse datasets, including CoQA (Reddy et al., 2019), TruthfulQA (Lin et al., 2022), and subsets of LongBench (Bai et al., 2024). Experimental results demonstrate that XQuant achieves an equivalent bit-width of less than 1.4-bit across various LLMs, outperforming existing methods such as KIVI-2bit (Liu et al., 2024b) and AsymKV-1.5bit (Tao et al., 2024). Notably, XQuant achieves comparable performance to full-precision baselines while offering a significantly improved trade-off between model performance and compression ratio.

2 Related Work

Two mainstream approaches for addressing KV cache challenges are Quantization and Eviction methods (Shi et al., 2024).

Quantization has emerged as a prominent technique for compressing large-scale models by mapping high-precision data to lower-precision formats (e.g., 16-bit, 8-bit, or even 4-bit integers). This significantly reduces memory footprints while maintaining acceptable levels of model performance. A substantial body of work focuses on quantizing model weights. AWQ (Lin et al., 2024) optimizes neural network weight quantization by dynamically adapting the bit-width based on the weights' significance. By retaining higher precision for more impactful weights and reducing precision for less critical ones, AWQ minimizes performance loss while achieving compression.

Another line of research concentrates on the quantization of the KV cache. KVQuant, introduced by Hooper et al. (2024), employs distinct quantization strategies for Keys and Values. It applies per-channel quantization to the Keys—particularly before Rotary Positional Embeddings (RoPE)—and per-token quantization to the Values, effectively managing outliers and minimizing RoPE-induced distortions. Similarly, MiKV (Yang et al., 2024c) introduces a mixed-precision KV-cache strategy that retains important KV pairs in high precision. By identifying critical KV pairs based on importance metrics, such as attention scores or contributions to the model's output, MiKV ensures context preservation and high-quality generation during inference. Concurrently, KIVI (Liu et al., 2024b) develops a tuning-free 2-bit KV cache quantization scheme, where the Key cache is quantized per-channel, and the Value cache is quantized per-token. Building on this, AsymKV (Tao et al., 2024) further combines 1-bit and 2-bit representations through an asymmetric and layer-wise quantization configuration, achieving a balance between precision and compression efficiency.

In contrast, some works simultaneously quantize both the model weights and the attention cache. For example, FlexGen (Sheng et al., 2023) introduces a high-throughput inference framework that applies group-wise 4-bit quantization to compress both the model weights and KV cache. FlexGen divides tensors into small groups, computes the minimum and maximum values within each group, and performs

asymmetric quantization. The resulting tensors are stored in 4-bit format and later dequantized to FP16 during computation, achieving a reduction in memory usage and I/O costs with minimal accuracy degradation. Despite the advancements of these methods, significant performance degradation remains a challenge when quantizing KV cache activations to extremely low-precision levels, particularly below 2-bit.

Eviction methods aim to discard unnecessary tokens during inference to reduce memory usage. StreamingLLM (Xiao et al., 2023) identifies the phenomenon of attention sinks, where initial tokens are retained to stabilize attention computations. StreamingLLM combines these attention sinks with a sliding window of recent tokens to introduce a rolling KV cache, effectively balancing memory efficiency and model performance. Building on this, SirLLM (Yao et al., 2024) uses token entropy to preserve critical tokens' KV cache and incorporates a memory decay mechanism to enhance LLMs' long-term memory while maintaining short-term reasoning abilities.

Other methods, such as H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024), dynamically identify and evict non-important tokens based on attention scores. PyramidKV (Cai et al., 2024; Yang et al., 2024a) observes that attention scores are more sparse in higher layers and accordingly allocates different memory budgets across layers. However, most existing KV eviction methods depend on attention scores to identify non-important tokens, which limits their compatibility with common optimizations like FlashAttention (Dao, 2023), reducing their practical usability.

Inter-layer redundancy Beyond the above intra-layer redundancy in KV caches, some studies have also explored the inter-layer redundancy. Some prior works (Wu and Tu, 2024; Sun et al., 2024; Brandon et al., 2024) investigate the potential of caching only partial layers of the KV cache, but all of them cannot be applied without additional training.

Compared to existing methods, we introduce XQuant with two key innovations: (1) A novel, simple yet effective Data-Free Calibration method that achieves superior compression performance even under ultra-low-bit settings, eliminating the need for additional calibration data. (2) Cross-Layer KV Cache Sharing that leverages previously overlooked quantization-enhanced layer similarities to achieve significant memory and computational savings. While prior work has studied layer representation similarities, our approach uniquely exploits the increased similarities post-quantization to enable effective ultra-low-bit compression.

3 XQuant

In this section, we present XQuant, a novel quantization framework for efficient KV cache compression. As illustrated in Figure 1, our framework introduces two key innovations: a data-free calibration technique that asymmetrical adjusts quantization parameters across layer pairs without additional calibration data, and a cross-layer KV cache compression mechanism that exploits enhanced similarity between adjacent quantized layers to reduce both computational and memory overhead.

3.1 Background

To formalize KV cache quantization, we consider a group of floating-point keys or values $\mathbf{X}$. The quantization process transforms $\mathbf{X}$ into three components: a B-bit quantized cache $\mathbf{X_Q}$, a scaling factor s, and a zero-point z (Liu et al., 2024b):

Quantization Phase:

$$z = min(\mathbf{X}), s = \frac{max(\mathbf{X}) - min(\mathbf{X})}{(2^B - 1)} \quad (1)$$

$$\mathbf{X_T} = (\mathbf{X} - z)/s, \mathbf{X_Q} = \lceil \mathbf{X_T} \rfloor \quad (2)$$

Dequantization Phase:

$$\hat{\mathbf{X}} = \mathbf{X_Q} * s + z \quad (3)$$

where $\mathbf{X}^*$ is the dequantized counterpart and $\lceil \cdot \rfloor$ is the rounding function. $\mathbf{X_T}$, the transformed matrix, is not cached but added in formula just for convenience.

Building upon this framework, prior works introduce various configurations to enhance performance. For example, Liu et al. (2024b) focuses on the element-wise distribution within the KV cache, adopting per-channel quantization for the key cache and per-token quantization for the value cache. Similarly, Tao et al. (2024) introduces layer-wise quantization configurations, employing asymmetric bit-widths for the key and value caches across different layers. While effective, these approaches often suffer from significant performance degradation under low-bit quantization settings, particularly around 2-bit precision. This limitation mo-

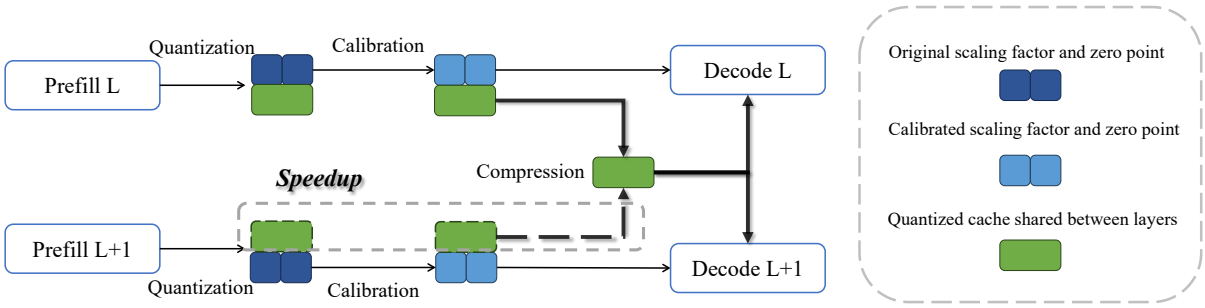

Figure 1: The illustration of XQuant workflow. We group the KV cache layer-wise in pairs. For every higher layer in a pair, we only compute and store the scaling factor and zero-point during quantization phase, and then fetch the quantized cache from the lower layer during dequantization phase.

tivates the need for further advancements in KV cache compression techniques.

3.2 Data-free Calibration

Achieving ultra-low-bit compression requires addressing the critical challenge of calibration. In this section, we propose a data-free calibration method that effectively preserves model performance while enabling aggressive compression ratios.

To analyze extreme quantization scenarios, we focus on 1-bit quantization where each parameter is constrained to a binary state. Formally, the rounding operation $\lceil \cdot \rfloor$ is defined as:

$$\lceil e \rfloor = \begin{cases} 0 & \text{if } e \in [0, 0.5], \\ 1 & \text{if } e \in (0.5, 1]. \end{cases} \tag{4}$$

where e denotes an element of the transformed matrix. For any bit-width B, this rounding operation maps values to a discrete set within $[0, 2^B - 1]$, where each original value is assigned to its nearest representative in the quantized space.

For 1-bit quantization, fixed representative values at endpoints (0 and 1) yield substantial quantization error. We therefore introduce a flexible mapping function that adaptively determines the quantization levels, formulated as:

$$f(e) = \begin{cases} \eta & \text{if } e \in [0, 0.5], \\ 1 - \eta & \text{if } e \in (0.5, 1]. \end{cases} \tag{5}$$

where $\eta \in [0, 0.5]$ serves as a calibration parameter for determining quantization tendancy. The improved mapping can be equivalently realized through adjustments to the scaling factor and zero-point, maintaining computational efficiency. We formalize the final data-free calibration approach as:

Consider a group of floating-point keys or values $\mathbf{X} \in \mathbf{R}^{gs}$, where gs stands for the group size. Note that $\mathbf{X} \in [min(\mathbf{X}), max(\mathbf{X})]^{gs} = [z, s * (2^B - 1) + z]^{gs}$, we can deduce:

$$\mathbf{X}_Q \in [0, 2^B - 1]^{gs} \tag{6}$$

from Equation 1 and Equation 2. If we choose $\eta * (2^B - 1)$ and $(1 - \eta) * (2^B - 1)$ generalized from Equation 5 as 2 endpoints, it is equivalent to calibrate the zero-point and scaling factor to $\hat{z}$ and $\hat{s}$, and then dequantize with them. Note that the dequantized matrix

$$\hat{\mathbf{X}} = \mathbf{X_Q} * \hat{s} + \hat{z} \in [\hat{s} * 0 + \hat{z}, \hat{s} * (2^B - 1) + \hat{z}]^{gs} \tag{7}$$

and the corresponding interval given by two endpoints:

$$[z + \eta s(2^B - 1), z + s(2^B - 1)(1 - \eta)] \tag{8}$$

By calculation we get the final operations for calibration:

$$\hat{z} = z + \eta s(2^B - 1), \hat{s} = (1 - 2\eta)s \tag{9}$$

We proposes the improved quantization scheme with this data-free calibration as follows:

Quantization Phase with Calibration:

$$z = min(\mathbf{X}), s = \frac{max(\mathbf{X}) - min(\mathbf{X})}{(2^B - 1)} \tag{10}$$

$$\mathbf{X_T} = (\mathbf{X} - z)/s, \mathbf{X_Q} = \lceil \mathbf{X_T} \rfloor \tag{11}$$

$$\hat{z} = z + \eta s(2^B - 1), \hat{s} = (1 - 2\eta)s \tag{12}$$

Dequantization Phase with Calibration:

$$\hat{\mathbf{X}} = \mathbf{X_Q} * \hat{s} + \hat{z} \tag{13}$$

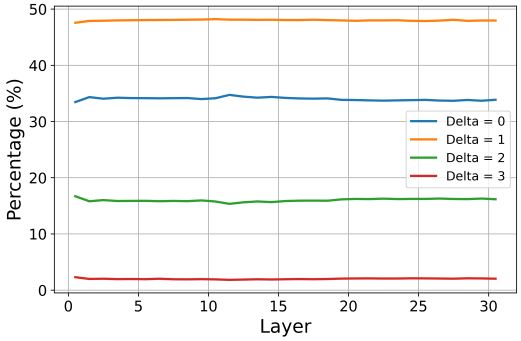

Figure 2: Layer-wise analysis of absolute differences between adjacent layers in quantized KV Cache matrices (2-bit quantization). Here, delta represents the absolute difference of quantized values between consecutive layers.

Method	Bit-width	MuSiQue	MFQA-Zh
Full Cache	16	9.98	28.89
KIVI	2	9.86	27.56
AsymKV-32/0	1.5	8.45	23.09
AsymKV-24/0	1.375	7.64	18.34

Table 1: Evaluation on LongBench based on AsymKV using Mistral-7b-instruct shows that the key cache is nearly impossible to quantized under 1-bit.

3.3 Cross-Layer Compression

3.3.1 Motivation

Building upon Tao et al. (2024)'s investigation of ultra-low-bit KV cache asymmetric quantization, our reproduction experiments on LongBench (Bai et al., 2023) with Llama (Touvron et al., 2023) demonstrate severe limitations of existing approaches, as shown in Table 1. We found that 1-bit asymmetric quantization of the key cache is practically infeasible, and even restricting 1-bit quantization to only the top 8 layers (AsymKV-24/0) leads to significant performance degradation. These findings suggest that Asym-KV primarily exploits the quantization potential of the value cache alone. This limitation motivates our development of a novel approach that simultaneously compresses both key and value caches, achieving an effective bit-width lower than previous methods. Given the impracticality of individual key cache quantization, we explore cross-layer compression methods to achieve equivalent ultra-low-bit quantization.

3.3.2 Analysis on Quantized KV Cache

To enable cross-layer compression, we first analyze the characteristics of quantized KV caches by examining inter-layer similarities. We hypothesize that significant redundancy between adjacent layers could create opportunities for more aggressive compression. Using the KIVI-2 framework (Liu et al., 2024b), we conduct preliminary experiments on the Mistral-7B-Instruct-v0.2 model (Jiang et al., 2023) with random samples from LongBench (Bai et al., 2023). Our analysis encompasses KV caches from both prefilling and decoding phases. For each pair of adjacent layers, we compute element-wise absolute differences between Value cache tensors across all attention heads, allowing us to characterize the distribution of inter-layer variations.

Under the 2-bit quantization scheme in KIVI-2, quantized cache values are restricted to {0, 1, 2, 3}, naturally constraining element-wise absolute differences to the same range. Our analysis, illustrated in Figure 1, reveals a striking pattern: over 80% of positions between adjacent layers exhibit minimal differences (0 or 1), while extreme differences (3) occur in less than 5% of positions. This pattern becomes even more pronounced in the 1-bit scenario, where mapping {0,1} to 0 and {2,3} to 1 maintains identical values in over 80% of positions between adjacent layers. These empirical findings demonstrate substantial redundancy in quantized KV caches between adjacent layers, suggesting significant potential for further compression.

3.3.3 Cross-Layer Compression

Leveraging these insights into inter-layer similarities, we propose a novel cross-layer compression method that decomposes KV caches into two components: shared quantized weights and layer-specific parameters. Specifically, adjacent layers share a common set of quantized value caches ($\mathbf{X_Q}$), while maintaining their individual scaling factors and zero-points for dequantization. This decomposition enables efficient compression by allowing each upper layer to reuse the quantized cache from its adjacent lower layer, while preserving layer-specific characteristics through unique calibration parameters. The approach significantly reduces both memory footprint and computational overhead while maintaining model performance.

In the implementation, for a model with L layers, we organize the layers into groups of size G. Within each group, KV caches are compressed using weighted averaging, where each layer l ($0 \leq l \leq L$) is assigned a weight γ_l, subject to the constraint $\sum \gamma_l = 1$

Formally, for two grouped layers l and $l+1$, the workflow quantization with cross-layer compres-

sion is utilized as follows:

Quantization Phase with Cross-Layer Compression and Calibration:

for $k = l, l+1, ...$

$$z^k = min(\mathbf{X^k}), s^k = \frac{max(\mathbf{X^k}) - min(\mathbf{X^k})}{(2^B - 1)}$$

$$\hat{z}^k = z^k + \eta s^k (2^B - 1), \hat{s}^k = (1 - 2\eta) s^k$$

$$\mathbf{X_Q} = \gamma_l \left\lceil \frac{\mathbf{X}^l - z^l}{s^l} \right\rfloor + \gamma_{l+1} \left\lceil \frac{\mathbf{X}^{l+1} - z^{l+1}}{s^{l+1}} \right\rfloor$$

Dequantization Phase with Cross-Layer Compression and Calibration:

$$\hat{\mathbf{X}}^l = \mathbf{X_Q} * \hat{s}^l + \hat{z}^l \tag{14}$$

$$\hat{\mathbf{X}}^{l+1} = \mathbf{X_Q} * \hat{s}^{l+1} + \hat{z}^{l+1} \tag{15}$$

We present the pseudo code for the whole workflow as shown in Appendix C.

3.3.4 Speedup through Cross-layer Compression

While our previous discussion introduced weighted averaging with weights γ_l and γ_{l+1} for compressing X_Q between layers l and $l+1$, we can further optimize the computation by setting $\gamma_l = 1$ for a chosen layer l, which consequently forces all other γ values within the group to zero. In this simplified configuration, X_Q is derived solely from the KV cache of the chosen layer in each group, substantially reducing computational overhead. Specifically,

$$\mathbf{X_Q} = \left\lceil \frac{\mathbf{X}^l - z^l}{s^l} \right\rfloor \tag{16}$$

As illustrated in Figure 1, this optimization eliminates the computations shown in the dashed line, effectively streamlining the process. Experimental results demonstrate that selecting the first layer within the group as the default layer selection achieves optimal performance.

4 Evaluation

4.1 Experimental Setup

Models We evaluate our XQuant on Llama-2-7b / Llama-2-7b-chat (Touvron et al., 2023) and Mistral-7B-v0.3 / Mistral-7B-instruct-v0.2 (Jiang et al., 2023).

Model	Method	CoQA	TruthfulQA
Mistral-7b	16-bit	68.02	32.09
	KIVI-2bit	67.78	32.17
	AsymKV-1.5bit	66.43	32.80
	XQuant-1.38bit	64.00	34.93
Llama2-7b	16-bit	63.88	30.77
	KIVI-2bit	63.25	33.92
	AsymKV-1.5bit	62.22	33.84
	XQuant-1.4bit	62.23	34.22

Table 2: Evaluation on LM-eval tasks with normal context length.

Tasks For the normal context length tasks, we choose CoQA (Exact match accuracy) and TruthfulQA (BLEU score) from LM-Eval (Gao et al., 2021). We also select several subsets from LongBench (Bai et al., 2023) for the long context length tasks, including HotpotQA (F1 score), 2WikiMultihopQA (F1 score), MuSiQue (F1 score), TREC (classification accuracy), TriviaQA (F1 score), SAMSum (Rouge-L) and PassageCount (Exact match accuracy). MultiFieldQA-Zh (F1 score) is selected for some ablation studies as well. In XQuant, we quantize the lower kq, vq layers of key and value cache into 2-bit, while quantizing others into 1-bit. We apply cross-layer compression from the kq th, vq th layer of key and value cache. All the configurations are summarized in Table 8.

Baselines and Implementations. We compare our framework with previous works, including original 16-bit floating implementation, KIVI-2 (Liu et al., 2024b) and AsymKV (Tao et al., 2024). All relevant configurations adhere as in KIVI, i.e., quantizing key cache per-channel and value cache per-token, and with a group size of 32 and a residual length of 128. We reproduce AsymKV based on the official implementation of KIVI, with a typical configuration (AsymKV-32/0) selected from the original paper, i.d., quantizing key cache into 2-bit and value cache into 1-bit, which is equivalent to 1.5-bit. We also choose a token eviction method (Yang et al., 2024b) for comparison on LongBench tasks as well, with a 40% KV cache setting. We set the maximum sequence length to 30000 for the Mistral model to conduct our experiments with a single NVIDIA GeForce RTX 3090 GPU (24GB), and 8192 for the Llama model as default. We do not consider SLERP (Shoemake, 1985; Liu et al., 2024a)) because of the incompatibility between rescale-recover operations and quantized cache.

Model	Method	Bit-width	HQA	2Wiki	MSQ	TREC	TQA	SAMS	PC	Avg
Mistral-7b-Ins	Full Cache	16	43.02	27.10	18.78	71.00	86.23	42.75	2.75	41.66
	PyramidInfer	/	35.08	23.92	16.90	62.00	85.06	41.45	1.04	32.55
	KIVI	2	41.96	26.08	18.13	71.00	86.00	43.70	2.78	41.38
	AsymKV	1.5	37.17	22.77	15.76	70.50	86.25	43.44	3.16	39.86
	XQuant	1.38	42.64	25.96	17.44	71.50	84.96	45.18	5.71	41.91
Llama2-7b-chat	Full Cache	16	30.09	26.48	9.98	63.00	84.19	41.22	4.50	37.07
	PyramidInfer	/	29.14	24.53	7.49	54.00	81.79	40.71	4.00	34.52
	KIVI	2	29.10	25.12	9.86	63.00	84.98	40.18	4.00	36.61
	AsymKV	1.5	27.75	24.82	8.45	62.00	84.21	41.22	2.75	35.89
	XQuant	1.4	29.16	25.14	9.69	62.50	84.57	40.01	4.00	36.44

Table 3: Evaluation of different KV cache compression methods on LongBench tasks.

4.2 Performance Comparison

LM-Eval Results The performance comparison between different compression methods is summarized in Table 2. We apply XQuant, KIVI and AsymKV to Mistral-7b-v0.3 and Llama2-7b. As shown in Table 2, with the lowest precision level, our XQuant surpass all other compression methods in TruthfulQA dataset while achieve a comparable performance in CoQA dataset with the lowest-precision quantization.

LongBench Results We evaluate XQuant on the LongBench benchmark using two widely adopted models: Mistral-7b-Instruct-v0.2 and Llama-2-7b-chat. As shown in Table 3, XQuant achieves significant improvements over other KV cache compression methods, particularly under ultra-low-bit settings.

For Mistral, XQuant achieves an average score of 41.91, surpassing AsymKV-1.5bit by a margin of 2.05 while maintaining a significantly lower bit-width of 1.38. Notably, XQuant retains performance comparable to the Full Cache baseline across datasets such as HQA (43.02 vs. 42.64). For Llama, compared to KIVI-2bit, XQuant improves results by 0.33 points while reducing the effective bit-width to 1.4. Additionally, compared to PyramidInfer, which sacrifices precision to reduce storage overhead, XQuant demonstrates clear advantages in maintaining high accuracy across tasks while achieving lower bit-width.

4.3 Ablation and Analysis

In this section, we conduct ablation studies in some randomly selected lightweight LongBench subsets.

Calibration Parameter As described in Section 3.2, there are parameters in our data-free calibration method, namely η_1 for 1-bit quantization and η_2 for 2-bit. We set η_1 = 0 or 0.2 and η_2 = 0 or 0.05 and deploy our calibration method on default KIVI-2bit and AsymKV-1.5bit to demonstrate its effectiveness. Note that $\eta = 0$ is equal to not deploying our calibration method. As shown in Table 4, these two quantization frameworks can perform better with our lightweight calibration method.

Method	Bit-width	η_1	η_2	MFQA-Zh
Full Cache	16	/	/	48.26
KIVI	2	/	0	42.27
KIVI	2	/	0.05	**44.34**
AsymKV	1.5	0	0	36.30
AsymKV	1.5	0	0.05	**41.28**
AsymKV	1.5	0.2	0	**42.78**
AsymKV	1.5	0.2	0.05	**43.81**

Table 4: The comparison using different quantization methods w/o our calibration method in MultiFieldQA-Zh and HotpotQA task from LongBench.

Method	Bit-width	γ_0	MuSiQue
Full Cache	16	/	18.78
KIVI	2	/	18.13
Flooring	1.63	/	16.79
Ceiling	1.63	/	16.36
Stochastic	1.63	/	17.65
Weighted Average	1.63	[0,1/6)	12.20
	1.63	(1/6,1/4)	14.05
	1.63	(1/4,1/2)	16.84
	1.63	(1/2,3/4)	17.32
	1.63	(3/4,5/6)	17.60
	1.63	(5/6,1]	17.32

Table 5: The comparison between different cross-layer compression method with group size $G = 2$, where γ_0, γ_1 stands for the coefficient in the weighted average $(\gamma_1 + \gamma_0 = 1)$.

Cross-Layer Compression Method Specifically, in the 2-bit quantization case where layers are grouped in pairs, Figure 2 shows that approximately 50% of the element-wise absolute differences are odd. This necessitates rounding opera-

Method	Bit-width	G	id	MSQ	MFQA-Zh
Full Cache	16	\	\	18.78	48.26
KIVI	2	\	\	18.13	42.27
XQuant	1.63	2	0	**17.32**	**37.44**
			1	12.20	20.48
		3	0	14.92	17.53
			1	16.97	37.37
			2	13.21	20.80
		4	0	14.82	23.53
			1	12.44	18.68
			2	16.12	35.48
			3	15.39	20.32

Table 6: The comparison of different group sizes G and chosen layer id within each group.

tions, such as floor rounding or stochastic rounding, when averaging values. We further explore the weighted average with a group size $G = 2$ and coefficients $\gamma_0, \gamma_1 = 1 - \gamma_0$, where γ_0 falls into six intervals listed in Figure 2. Notably, when $\gamma_0 \in [0, 1/6)$ or $\gamma_0 \in (5/6, 1]$, the operation simplifies to directly sharing the quantized cache.

We evaluate KIVI-2 on Mistral-7B-Instruct-v0.2 without the proposed calibration methods starting from the 8-th layer, using a group size of 2. As summarized in Table 5, the degraded compression methods ($\gamma_0 \in [0, 1/6) \cup (5/6, 1]$) avoid redundant unpacking and packing operations seen in the workflow of (Liu et al., 2024b), which rounds quantized integers into 32-bit. These methods strike a better balance between efficiency and memory usage. In this ablation study, we demonstrate that one degraded compression method is sufficient for effective performance.

As shown in Table 5, the proposed degraded compression operation demonstrates its effectiveness in maintaining sufficient information for model performance, particularly when the coefficient γ_0 falls within the range $(5/6, 1]$. This configuration effectively allows odd-numbered layers to reuse the quantized cache from the preceding even-numbered layers without requiring additional quantization or storage overhead for the odd layers. We adopt this weighted averaging strategy across all subsequent experiments due to its favorable balance between computational efficiency and information preservation.

In summary, our experiments highlight that cross-layer KV cache sharing, particularly with optimized weighted averaging, provides a computationally efficient and memory-saving alternative that achieves competitive performance under ultra-low-bit settings.

Method	Bit-width	TREC	SAMS
Full Cache	16	71.00	42.75
KIVI	2	71.00	43.70
AsymKV	1.5	70.50	43.44
XQuant	1.375	71.50	45.18
XQuant	1.28	68.50	39.84
XQuant	1.15625	68.50	39.47

Table 7: The comparison of different configurations under extremely-low compression ratio.

Group Size. After optimizing the cross-layer compression method, another factor is the group size. To investigate the effects of layer grouping, we partition the 32 layers into groups based on different grouping strategies. The idx parameter means we only store and share the quantized cache in the idx-th layer in each group. We set all configurations under the same compression ratio, namely keep all layers in key cache and 20 layers in value cache based on KIVI-2bit framework, using Mistral-7b-instruct-v0.2. As shown in Table 6, the model achieves the best performance with the configuration of group_size = 2 and idx = 0.

Extreme Compression. To explore the limits of low-bit quantization, we evaluate XQuant on the TREC and SAMSum subsets from LongBench using Mistral-7b-instruct-v0.2 under extremely low compression ratios. As shown in Table 7, XQuant maintains competitive accuracy even at an ultra-low 1.15625-bit setting, achieving over 90% of the full-precision floating-point baseline performance.

5 Conclusion

To alleviate the growing memory overhead in LLM inference, we propose XQuant, a plug-and-play framework that quantize KV cache at an extreme compression ratio. Based on our observations on classical training-free quantization and the distributions of quantized integers, we propose a data-free calibration method and a compute-efficient cross-layer compression method. Extensive experiments show that XQuant achieves the state-of-the-art trade-offs between performance degradation and compression ratio. Integrating these two novel methods, our XQuant achieves comparable performance with full-precision baseline under 1.4-bit quantization, and still maintains competitive performance for some tasks under an extremely 1.16-bit quantization.

Limitation

While XQuant achieves efficient ultra-low-bit KV cache quantization, it faces challenges in parameter selection, which remains complex and highly task- and dataset-dependent. The performance of quantization is sensitive to hyperparameters, such as calibration coefficients, requiring manual tuning for different scenarios. In the future, research should focus on automating parameter selection through adaptive methods, such as dynamic calibration or learning-based approaches, to ensure optimal performance across diverse tasks and datasets with minimal human intervention.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore. Association for Computational Linguistics.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3119–3137. Association for Computational Linguistics.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

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

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

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 10:8–9.

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

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

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

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3214–3252. Association for Computational Linguistics.

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

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *ArXiv*, abs/2402.02750.

Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. 2024. Comprehensive cognitive llm agent for smartphone gui automation. *arXiv preprint arXiv:2402.11941*.

Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. 2023. Llm is like a box of chocolates: the non-determinism of chatgpt in code generation. *arXiv preprint arXiv:2308.02828*.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. Coqa: A conversational question answering challenge. *Trans. Assoc. Comput. Linguistics*, 7:249–266.

Nikhil Sharma, Q Vera Liao, and Ziang Xiao. 2024. Generative echo chamber? effect of llm-powered search systems on diverse information seeking. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–17.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. 2024. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*.

Ken Shoemake. 1985. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254.

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

Qian Tao, Wenyuan Yu, and Jingren Zhou. 2024. Asymkv: Enabling 1-bit quantization of kv cache with layer-wise asymmetric quantization configurations. *arXiv preprint arXiv:2410.13212*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

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

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024a. Pyramidinfer: Pyramid KV cache compression for high-throughput LLM inference. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 3258–3270. Association for Computational Linguistics.

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

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

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. 2024d. Kvsharer: Efficient inference via layer-wise dissimilar kv cache sharing. *arXiv preprint arXiv:2410.18517*.

Yao Yao, Zuchao Li, and Hai Zhao. 2024. Sirllm: Streaming infinite retentive llm. *arXiv preprint arXiv:2405.12528*.

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

A Compression Ratio Analysis

Formally, let b, h, s, d be the batch size, the number of heads in GQA (Ainslie et al., 2023), the sequence length and the dimension per head. The original l layers of KV cache occupies $2l * bhsd * 16\ bit$, which equals to $2l * n * 16\ bit$ if we set $n = bhsd$ for convenience.

Consider a typical KV cache quantization scheme ((Liu et al., 2024b)). If we quantize all l layers of key cache and value cache into b-bit, the quantized KV cache memory usage is $2l * n * b\ bit$. (Tao et al., 2024) uses a asymmetrical configurations for key and value caches across different layers. In their paper, Asym-l_k/l_v means quantizing the initial l_k layers of key cache and l_v of value cache into 2-bit, and quantizating 1-bit for others. So the quantized KV cache memory usage is $(2 * l_k + (32 - l_k) + 2 * l_v + (32 - l_v)) * n\ bit$. For example, Asym-1.5bit stands for Asym-32/0 in our paper, which can be calculated to $3l * n\ bit$ and can be equivalently considered as a 1.5-bit symmetrical quantization for better understanding of the compression ratio. The related parameters in XQuant are kq, vq, km, vm. The equivalent bit-width B can be expressed as follows: $B = ((32 - max(kq, km))/2 + (max(kq, km) - min(kq, km)) * 1 + (max(kq, km) + min(kq, km)) * 2 + (32 - max(vq, vm))/2 + (max(vq, vm) + min(vq, vm)) * 1 + (max(vq, vm) + min(vq, vm)) * 2)/64$,

B Configurations

The Configurations of XQuant in our main experiments are summarized in Table 8

C XQuant Pseudo Code

The pseudo code for the whole workflow can be found in Algorithm 1 and 2.

Model	Dataset	kq	vq	km	vm	eta1	eta2
Mistral-7b-v0.3	CoQA	30	2	32	16	1/6	0.045
	TruthfulQA	30	2	32	16	0	0
Mistral-7b-instruct-v0.2	HQA	30	2	32	16	1/6	0.045
	2Wiki	30	2	32	16	1/6	0.09
	MSQ	32	0	32	16	1/6	0
	TREC	30	2	32	16	1/6	0
	TQA	30	2	32	16	1/6	0.09
	SAMS	30	2	32	16	0	0
	PC	32	0	32	16	0	0.045
Llama2-7b	CoQA	28	0	32	28	0	0
	TruthfulQA	28	0	32	28	1/3	0
Llama2-7b-chat	HQA	32	0	32	20	1/6	0
	2Wiki	28	0	32	28	1/3	0
	MSQ	28	0	32	28	1/3	0
	TREC	32	0	32	20	1/6	0
	TQA	32	0	32	20	1/6	0
	SAMS	32	0	32	20	0	0
	PC	32	0	32	20	1/3	0.045

Table 8: The configurations of our main experiments.

Algorithm 1: XQuant Procedure

Input : kq, vq, km, vm, $\eta[2]$
Output : Optimized Quantized Cache

for $l \leftarrow 0$ ***to*** 31 **do**
  if $l < vm$ ***or*** $l \bmod 2 == 0$ **then**
    KeyCache$[l] \leftarrow$ **Quantize**$\big(X_k^l,\ 2$ **if** $l < kq$ **else** $1\big)$
  else
    KeyCache$[l] \leftarrow$ **PseudoQuantize**$\big(X_k^l,\ 2$ **if** $l < kq$ **else** $1\big)$
  if $l < vq$ ***or*** $l \bmod 2 == 0$ **then**
    ValueCache$[l] \leftarrow$ **Quantize**$\big(X_v^l,\ 2$ **if** $l < vq$ **else** $1\big)$
  else
    ValueCache$[l] \leftarrow$ **PseudoQuantize**$\big(X_v^l,\ 2$ **if** $l < vq$ **else** $1\big)$
for $l \leftarrow 0$ ***to*** 31 **do**
  if $l < km$ ***or*** $l \bmod 2 == 0$ **then**
    DequantizedKey $\leftarrow$ **Dequantize**$\big($KeyCache$[l][0]$, KeyCache$[l][1]$, KeyCache$[l][2]\big)$
  else
    DequantizedKey $\leftarrow$ **Dequantize**$\big($KeyCache$[l-1][0]$, KeyCache$[l-1][1]$, KeyCache$[l][2]\big)$
  if $l < vm$ ***or*** $l \bmod 2 == 0$ **then**
    DequantizedValue $\leftarrow$ **Dequantize**$\big($ValueCache$[l][0]$, ValueCache$[l][1]$, ValueCache$[l][2]\big)$
  else
    DequantizedValue $\leftarrow$ **Dequantize**$\big($ValueCache$[l-1][0]$, ValueCache$[l-1][1]$, ValueCache$[l][2]\big)$

Algorithm 2: Supporting Functions

Function `PseudoQuantize`(X, n_bits)**:**
  $zero\_point \leftarrow \min(X)$ `// Find the minimum value of` X;
  $scaling\_factor \leftarrow \frac{\max(X)-\min(X)}{2^{n\_bits}-1}$ `// Calculate scaling factor;`
  return **Calibrate**$(zero\_point, scaling\_factor, n\_bits)$, None;
Function `Quantize`(X, n_bits)**:**
  $zero\_point \leftarrow \min(X)$;
  $scaling\_factor \leftarrow \frac{\max(X)-\min(X)}{2^{n\_bits}-1}$;
  $quantized\_cache \leftarrow$ **round** $\left(\frac{X-zero\_point}{scaling\_factor}\right)$ `// Round to nearest quantized value;`
  return **Calibrate**$(zero\_point, scaling\_factor, n\_bits)$, $quantized\_cache$;
Function `Dequantize`($zero\_point$, $scaling\_factor$, $quantized\_cache$)**:**
  return $quantized\_cache \cdot scaling\_factor + zero\_point$ `// Reconstruct original value;`
Function `Calibrate`($zero\_point$, $scaling\_factor$, n_bits)**:**
  $zero\_point\_cali \leftarrow zero\_point + scaling\_factor \cdot \eta[n\_bits]$ `// Adjust zero point based on` η;
  $scaling\_factor\_cali \leftarrow scaling\_factor \cdot \big(1 - 2 \cdot \eta[n\_bits]\big)$ `// Adjust scaling factor based on` η;
  return $zero\_point\_cali$, $scaling\_factor\_cali$ `// Return calibrated values;`