XQuant: Achieving Ultra-Low Bit KV Cache Quantization 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, particularly due to KV cache growth during long-text understanding and generation, present significant challenges for deployment in resource-constrained environments. Quantization has emerged as a promising solution to reduce memory consumption while preserving historical information. We propose XQuant, a training-free and plug-and-play framework that achieves ultra-low equivalent bit-width KV cache quantization. XQuant introduces two key innovations: a computationally negligible data-free calibration method and cross-layer KV cache compression, enabling quantization to sub-1.4 bits. Extensive experiments on TruthfulQA and LongBench demonstrate that XQuant outperforms state-of-the-art methods (e.g., KIVI-2bit and AsymKV-1.5bit) by achieving lower bit-width while maintaining superior performance, establishing a better trade-off between memory efficiency and model accuracy.

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 KV cache quantization methods, due to inherent architectural constraints, fail to mitigate the severe performance degradation 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 Compression:** We observe enhanced KV cache similarity between adjacent layers after quantization - a previously overlooked phenomenon. This enables effective cross-layer compression, where the quantized KV cache of one layer is shared across subsequent layers, significantly reducing computational and memory costs. Meanwhile, a subset of layer-specific parameters is preserved to retain the unique characteristics of each layer, ensuring minimal loss of 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 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. 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 better trade-off between precision and compression ratio.

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. We further clarify the key differences and highlight our contributions in Appendix G.

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 compression 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 quantization-enhanced similarities 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 asymmetrically adjusts quantization parameters without additional calibration data, and a cross-layer KV cache compression mechanism that leverages the similarity of quantized caches between adjacent layers to effectively 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 zero-point z, and a scaling factor s (Liu et al., 2024b):

Quantization Phase:

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

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

Dequantization Phase:

$$\hat{\mathbf{X}} = \mathbf{X_Q} * s + z \tag{3}$$

where $\mathbf{X}^*$ is the dequantized counterpart and $\lceil \cdot \rfloor$ is the rounding function. $\mathbf{X_T}$, the transformed matrix, is not explicitly cached but is introduced as an intermediate variable to facilitate subsequent mathematical derivations.

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

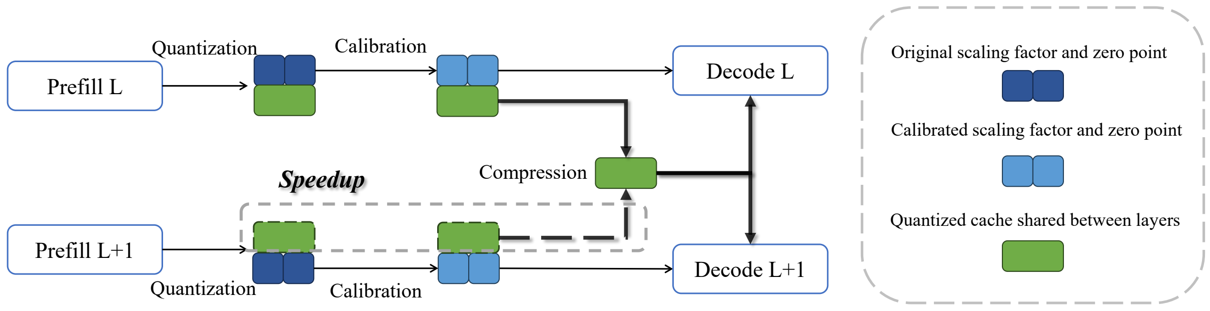

Prefill L
Quantization
Calibration
Decode L
Speedup
Compression
Prefill L+1
Quantization
Calibration
Decode L+1
Original scaling factor and zero point
Calibrated scaling factor and zero point
Quantized cache shared between layers

Figure 1: The illustration of XQuant workflow. XQuant partitions the KV cache into layer-wise pairs. For every higher layer in a pair, XQuant only computes and stores the scaling factors and zero-points during quantization phase, and then fetches the quantized cache from the lower layer during dequantization phase.

3.2 Data-free Calibration

Since existing quantization methods often experience significant performance degradation at 2-bit precision, achieving ultra-low-bit compression first requires bridging this performance gap. In this section, we propose a data-free calibration method that effectively preserves model performance, enabling more aggressive compression ratios.

To analyze extreme quantization scenarios, we start with 1-bit quantization where each parameter is constrained to a binary state. Formally, the round-to-nearest 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. As shown in Figure 2(a), fixed representative values at endpoints (0 and 1) yield substantial quantization error for 1-bit quantization. We therefore introduce a relaxed-constraint 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 tendencies. We extend this formulation to the general case of B-bit quantization and denote the corresponding parameter as η_B.

We relax the constraint that quantized values must be integers and apply fake quantization as a preliminary experiment. Table 7 shows that using this constraint-relaxed mapping function improves model performance, validating our proposed insight.

However, storing floating-point numbers as so-called quantized caches is impractical, as shown in Figure 2(b). To address the aforementioned problem, we establish an equivalent implementation, with the mathematical proof provided below. We formalize the final data-free calibration approach as:

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

$$\mathbf{X_Q} \in [0, 2^B - 1]^g \tag{6}$$

from Equation 1 and Equation 2. If we choose $\eta * (2^B - 1)$ and $(1 - \eta) * (2^B - 1)$ generalized from Equation 5 as two 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

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

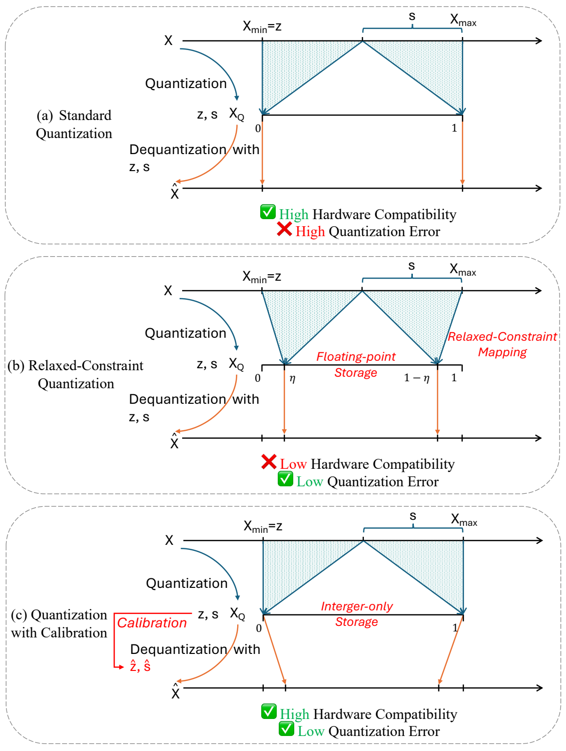
and the corresponding interval given by two endpoints:

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

By calculation we get the final operations for calibration:

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

As shown in Figure 2(c), we propose the improved quantization scheme with this data-free



(a) Standard Quantization
X
X_min=z
s
X_max
Quantization
z, s X_Q
0
1
Dequantization with z, s
X̂
✅ High Hardware Compatibility
❌ High Quantization Error
(b) Relaxed-Constraint Quantization
Floating-point Storage
Relaxed-Constraint Mapping
η
1 − η
❌ Low Hardware Compatibility
✅ Low Quantization Error
(c) Quantization with Calibration
Calibration
Dequantization with ẑ, ŝ
Interger-only Storage
✅ High Hardware Compatibility
✅ Low Quantization Error

Figure 2: The illustration of the proposed data-free calibration method.

calibration as follows:

Quantization Phase with Calibration:

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

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

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

Dequantization Phase with Calibration:

$$\hat{\mathbf{X}} = \mathbf{X_Q} * \hat{s} + \hat{z} \quad (13)$$

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 Mistral (Jiang et al., 2023) demonstrate severe limitations of existing approaches, as shown in Table 8.

We found that 1-bit asymmetric quantization of the key cache is practically infeasible. Even when restricting 1-bit quantization to the top 8 layers

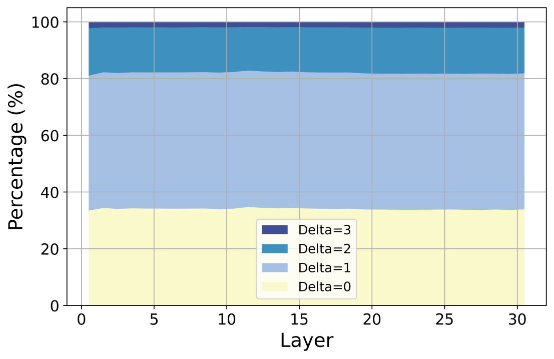


Figure 3: Layer-wise analysis of absolute differences between adjacent layers in quantized KV Cache matrices. Here, delta represents the absolute difference of quantized values between consecutive layers.

(AsymKV-24/32), significant performance degradation occurs. Given the limitations of further key cache quantization, we turn to cross-layer compression techniques as a viable alternative to achieve comparable ultra-low-bit quantization without compromising performance.

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).

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 3, 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 Compression Algorithm

Leveraging these insights into inter-layer similarities, we propose a novel cross-layer compression method that decomposes KV caches into two components: shared quantized caches 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 layer to reuse the merged cache from its group, while preserving the layer-specific characteristic through its unique quantization parameters, namely zero-points and scaling factors.

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 every layer l in a group G, the quantization workflow with cross-layer compression and calibration is utilized as follows:

Quantization Phase with Cross-Layer Compression and Calibration:

$$\forall l \in \mathbf{G},$$

$$z_l = min(\mathbf{X}_l),\ s_l = \frac{max(\mathbf{X}_l) - min(\mathbf{X}_l)}{(2^B - 1)}$$

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

$$\mathbf{X_Q} = \sum_{l \in \mathbf{G}} \gamma_l \left\lceil \frac{\mathbf{X}_l - z_l}{s_l} \right\rfloor$$

Dequantization Phase with Cross-Layer Compression and Calibration:

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

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

3.3.4 Speedup through Cross-layer Compression

While our previous discussion introduced weighted averaging with the weight γ for compressing $\mathbf{X_Q}$ within a group, we can further optimize the computation by setting $\gamma_k = 1$ for a chosen dominant layer k, which consequently forces all other γ values within the group to zero. In this accelerated configuration, each subordinate layer only needs to compute and store its own scaling factors and zero-points, significantly reducing computational overhead. Specifically,

$$\mathbf{X_Q} = \left\lceil \frac{\mathbf{X}_k - z_k}{s_k} \right\rfloor$$

Model	Method	Bit-width	TruthfulQA
Mistral-7b	Full Cache	16	32.09
	KIVI	2	32.17
	AsymKV	1.5	32.80
	XQuant	1.38	**34.93**
Llama2-7b	Full Cache	16	30.77
	KIVI	2	33.92
	AsymKV	1.5	33.84
	XQuant	1.4	**34.22**

Table 1: Evaluation on TruthfulQA task with normal context length.

As illustrated in Figure 1, this optimization eliminates the computations shown in the dashed line, effectively streamlining the process. Experimental results show that selecting the first layer within the group as the dominant layer yields optimal performance, as demonstrated in Table 4 and Table 5.

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).

Tasks. For the normal context length task, we choose 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.

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 equiva-

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	**42.90**	**26.65**	17.44	**71.50**	84.50	**45.18**	**5.71**	**41.98**
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	**29.21**	**25.56**	9.69	62.50	84.57	40.01	**4.00**	36.51

Table 2: Evaluation of different KV cache compression methods on LongBench tasks.

lent 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.

Method	Bit-width	η_1	η_2	MFQA-Zh
Full Cache	16	/	/	48.26
KIVI	2	/	0	42.27
AsymKV	1.5	0	0	36.30
XQuant	1.375	0	0	37.20
		0	0.05	**40.32**
		0.2	0	**41.98**
		0.2	0.05	**44.20**

Table 3: Ablation study on the effect of data-free calibration in XQuant on the MultiFieldQA-Zh benchmark from LongBench.

4.2 Performance Comparison

LM-Eval Results. Table 1 presents the evaluation of different quantization methods on the TruthfulQA task with a standard context length. XQuant not only achieves competitive performance but surpasses the full cache baseline, with a TruthfulQA score of 34.93 on Mistral-7b and 34.22 on Llama2-7b, outperforming all other methods at significantly lower bit-widths. These results highlight that XQuant provides superior performance in conventional context length settings.

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 2, XQuant achieves significant improvements over other KV cache compression methods, particularly under ultra-low-bit settings.

In all datasets of LongBench, XQuant achieves performance comparable to the full cache baseline while reducing bit-width by 31% compared to KIVI-2bit. Moreover, XQuant outperforms AsymKV on nearly all datasets while simultaneously reducing bit-width by 8% relative to AsymKV. 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. Table 3 presents an ablation study on the impact of data-free calibration in XQuant on the MultiFieldQA-Zh benchmark. The results indicate that applying calibration ($\eta_1 \neq 0$ or $\eta_2 \neq 0$) significantly improves XQuant's performance, reducing the performance gap with the full cache baseline.

Cross-Layer Compression Method. 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 derived in Appendix F. Notably, when $\gamma_0 \in [0, 1/6)$ or $\gamma_0 \in (5/6, 1]$, the operation is optimized to directly sharing the quantized cache. We evaluate KIVI-2 on Mistral-7B-Instruct-v0.2 without our proposed calibration methods starting from the 8-th layer. As summarized in Table 4, the accelerated compression methods ($\gamma_0 \in [0, 1/6) \cup (5/6, 1]$) avoid redundant operations seen in the workflow of Liu et al., 2024b,

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
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 4: 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 $(\gamma_1 + \gamma_0 = 1)$.

Method	Bit-width	G	k	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 5: The comparison of different group sizes G and selection indices k within each group, where XQuant is employed without the calibration step for a clearer analysis.

which rounds quantized integers into floating-point numbers. As shown in Table 4, the accelerated compression operation demonstrates its effectiveness in maintaining sufficient information for model performance, particularly when $\gamma_0 \in (5/6, 1]$. 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 odd-numbered layers.

We adopt this accelerated compression strategy across all experiments due to its favorable balance between computational efficiency and information preservation.

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 parameter k indicates that we store and share the quantized cache only in the k-th layer of 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 5, the model achieves the best performance with the configuration of $G = 2$ and $k = 0$.

Method	Bit-width	TREC	SAMS
Full Cache	16	71	42.75
KIVI	2	71	43.7
AsymKV	1.5	70.5	43.44
AsymKV	1.375	69.5	42.76
XQuant	1.375	**71.5**	**45.18**
AsymKV	1.28	58.5	37.41
XQuant	1.28	**68.5**	**39.84**
AsymKV	1.15625	41	23.47
XQuant	1.15625	**68.5**	**39.47**

Table 6: The comparison of different configurations under extremely-low compression ratio.

Performance-Compression Trade-offs. Table 6 evaluates the trade-offs between bit-width reduction and performance degradation across different quantization methods. As shown in Table 6, XQuant consistently outperforms other methods at the same bit-width, achieving higher scores on both TREC and SAMS benchmarks. Notably, even at an extremely low bit-width of 1.15625, XQuant preserves a significant portion of the model's performance, maintaining a TREC score of 68.5 compared to the full-cache baseline of 71. These results demonstrate that XQuant effectively balances performance retention and compression, achieving state-of-the-art trade-offs in ultra-low-bit KV cache quantization.

5 Conclusion

To alleviate the growing memory overhead in LLM inference, we propose XQuant, a plug-and-play framework that quantizes 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 state-of-the-art trade-offs between performance degradation and compression ratio, without sacrificing computational efficiency. 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 around an extremely 1.16-bit quantization.

Limitation

While XQuant demonstrates promising results across a range of representative models and benchmarks, future work may further validate its robustness and generalizability by extending evaluations to larger-scale models and more diverse downstream scenarios. Such expansion, given sufficient time and computational resources, would help reinforce the applicability of XQuant in broader real-world deployments.

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*.

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.

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 7: The comparison using different quantization methods with and without our calibration method in MultiFieldQA-Zh tasks from LongBench.

A Preliminary Study on Relaxed-Contraint Mapping

As demonstrated in Figure 2, the traditional quantization workflow faces higher quantization error in low-bit scenarios. In Section 3.2, we propose a flexible mapping to mitigate the quantization error in this aspect. Moreover, to provide empirical evidence supporting the effectiveness of the flexible mapping in the proposed calibration method, we employ its generalized form and conduct a preliminary study on the default KIVI-2bit and AsymKV-32/0 configurations. We extend this approach to a generalized B-bit quantization mechanism, where η_B serves as the corresponding parameter. Notably, when $\eta_B = 0$, the B-bit quantization operates without the flexible mapping.

The results in Table 7 demonstrate that incorporating the flexible mapping function enhances model performance across different quantization settings.

B Preliminary Experiment on Layer-Wise Asymmetric Quantization

In the existing method (Tao et al., 2024), the KV cache for each layer is quantized using either 1-bit or 2-bit precision. A straightforward strategy to maximize the compression ratio is to apply 1-bit quantization to a greater number of layers.

However, a significant bottleneck arises, as it is nearly impossible to quantize the key cache at 1-bit precision without compromising performance. As shown in Table 8, further compression by increasing the number of 1-bit quantized key cache layers is not feasible, as it leads to substantial performance degradation. This observation motivates us to explore alternative compression methodologies.

Method	**Bit-width**	**# Key Layers in 1-bit**	**MFQA-Zh**
Full Cache	16	/	48.26
KIVI (32/32)	2	0	42.27
AsymKV-24/32	1.875	8	37.10
AsymKV-16/32	1.75	16	21.36
AsymKV-8/32	1.625	24	13.16
AsymKV-0/32	1.5	32	7.66

Table 8: Evaluation on LongBench based on AsymKV shows that the key cache is nearly impossible to quantized under 1-bit.

C Equivalent Bit-width 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, and vm. The equivalent bit-width B can be expressed as follows: $B = ((32 - max(kq, km))/2 + (max(kq, km) - min(kq, km)) + (max(kq, km) + min(kq, km)) * 2 + (32 - max(vq, vm))/2 + (max(vq, vm) + min(vq, vm)) + (max(vq, vm) + min(vq, vm)) * 2)/64$.

In the classical configuration in our paper, $kq = 30$, $vq = 2$, $km = 32$, and $vm = 16$, in key cache we apply 2-bit quantization to the layers $[0, kq)$ and 1-bit quantization to the layers $[kq, 32)$, and cross-layer compression to the layers $[km, 32)$. The value cache is processed in the same manner. Therefore, the equivalent bit-widths of the key and value caches are computed as follows:

$$B_k = \frac{(32 - 30) + 30 * 2}{32} = 1.9375$$

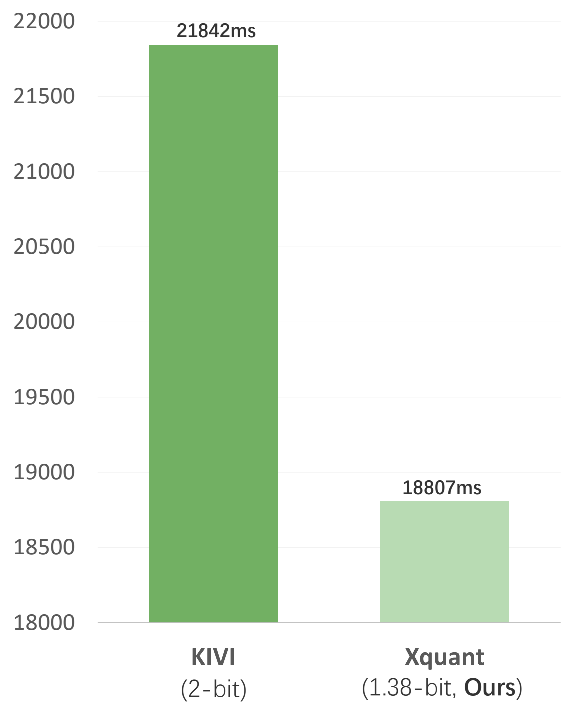


Figure 4: Comparison of Execution Time.

$$B_v = \frac{(32-16)/2 + (16-2) + 2*2}{32} = 0.8125$$

The average bit-width is therefore 1.375, which appears as 1.38 in most parts of this paper. More parameter sets used in our experiments are listed in Appendix H.

D Efficiency analysis

Using Mistral-7B as an example, we theoretically analyze the computational cost of our two key improvements. During the **calibration** step, generating each token incurs only 64 additional floating-point multiplications and 32 additions (Equation 12), which are negligible in practice. Moreover, as described in Section 3.3.4, the **cross-layer compression** step optimizes efficiency by skipping certain parts of the quantization process (Equation 2).

To evaluate inference efficiency, we adopt the same experimental setup as implemented in KIVI's repository, using a batch size of 16, a prompt length of 1024, and an output length of 128. As shown in Figure 4, XQuant, by leveraging its unique speedup mechanism, demonstrates competitive inference efficiency.

E Hyperparameter

The related parameters in XQuant are kq, vq, km, and vm. 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 km th, vm th layer of key and value cache. All the configurations are summarized in Table 10.

As demonstrated in Table 9, additional experiments on the Mistral-7B-Instruct model using the LongBench benchmark show that XQuant, with a fixed $\eta_1 = 1/6$ and $\eta_2 = 0.045$, consistently delivers strong performance as well. These results suggest that this fixed set of hyperparameters are robust and can generalize effectively across different datasets. Therefore, task-specific hyperparameter tuning is superior but not necessary, and the method can achieve reliable performance with a fixed, pre-selected set of hyperparameters.

F Cross-Layer Compression Strategy

Under 2-bit quantization, the values in the KV cache are restricted to the discrete integer set $\{i \in \mathbf{Z} \mid 0 \leq i \leq 3\}$. Therefore, a rounding operation is required after weighted averaging. If standard rounding-to-nearest is applied, the range of γ_0 can be divided into six disjoint intervals, as summarized in Table 4. The derivation is as follows:

Let e_0 and e_1 denote the B-bit quantized values at the same position in adjacent layers of $\mathbf{X_Q}$. Then the merged value e_m after cross-layer compression is computed as:

$$\begin{aligned} e_m &= \left\lfloor \frac{\gamma_0 e_0 + \gamma_1 e_1}{\gamma_0 + \gamma_1} \right\rceil \\ &= \lfloor \gamma_0 e_0 + (1-\gamma_0) e_1 \rceil \\ &= e_1 + \lfloor \gamma_0 (e_0 - e_1) \rceil . \end{aligned}$$

Without loss of generality, assume $e_0 \geq e_1$ and define $\delta = e_0 - e_1 \geq 0$. Then we have:

$$e_m = e_1 + \lfloor \gamma_0 \delta \rceil , \tag{14}$$

where $\gamma_0 \in [0,1]$ and $\delta \in \mathbf{Z} \cap [0,3]$. Since $\gamma_0\delta \in [0,\delta]$, the rounding term $\lfloor \gamma_0\delta \rceil$ in Eq. 14 can only take $\delta + 1$ discrete values. Let $\lfloor \gamma_0\delta \rceil = c$, where $c \in \mathbf{Z} \cap [0,\delta]$. Then:

$$\gamma_0\delta \in \left(c - \frac{1}{2}, c + \frac{1}{2}\right) \cap [0,\delta], \tag{15}$$

Method	Bit-width	Hyperparameters	HQA	2Wiki	MSQ	TREC	TQA	SAMS	PC	Avg
Full Cache	16	/	43.02	27.10	18.78	71.00	86.23	42.75	2.75	41.66
AsymKV	1.5	/	37.17	22.77	15.76	70.50	86.25	43.44	3.16	39.86
XQuant	1.38	Task-specific	42.90	26.65	17.44	71.50	84.50	45.18	5.71	41.98
XQuant	1.38	**Static**	**42.64**	**25.16**	**16.91**	**70.50**	**84.50**	**42.64**	**4.57**	**40.99**

Table 9: Evaluation of different KV cache compression methods using static hyperparameters setting.

which yields the following constraint for γ_0, when $\delta > 0$:

$$\gamma_0 \in \left(\frac{c - 1/2}{\delta}, \frac{c + 1/2}{\delta}\right) \cap [0, 1]. \qquad (16)$$

We now enumerate all valid combinations of δ and c from Equation 16:

- $\delta = 0$: Only one possible value exists; trivial case omitted.
- $\delta = 1$:
    - $c = 0$: $\gamma_0 \in [0, 1/2)$
    - $c = 1$: $\gamma_0 \in (1/2, 1]$
- $\delta = 2$:
    - $c = 0$: $\gamma_0 \in [0, 1/4)$
    - $c = 1$: $\gamma_0 \in (1/4, 3/4)$
    - $c = 2$: $\gamma_0 \in (3/4, 1]$
- $\delta = 3$:
    - $c = 0$: $\gamma_0 \in [0, 1/6)$
    - $c = 1$: $\gamma_0 \in (1/6, 1/2)$
    - $c = 2$: $\gamma_0 \in (1/2, 5/6)$
    - $c = 3$: $\gamma_0 \in (5/6, 1]$

Collectively, this yields six effective intervals of γ_0, as summarized in Table 4.

G Comparison with Other Cross-Layer Compression Methods

Several prior works have explored inter-layer redundancy from different perspectives. To eliminate potential confusion, we clarify several key distinctions and highlight innovations as follows: (**a**) Most existing methods compute KV caches at a subset of layers. However, these approaches require additional training steps and, in some cases, even full retraining, significantly limiting scalability. In contrast, XQuant is designed as a plug-and-play solution that leverages deeper insights to enable effective redundancy reduction without any additional training. (**b**) XQuant is the only method that explicitly considers inter-layer redundancy through the lens of quantization. After quantization, the KV cache is decomposed into three components: the quantized cache, zero-points, and scaling factors. We demonstrate that the quantized cache, consisting solely of integers, exhibits substantial inter-layer similarity. Meanwhile, the zero-points and scaling factors, which require minimal storage, are retained individually to preserve per-layer characteristics without being compressed. (**c**) MiniCache (Liu et al., 2024a) is another training-free method that primarily introduces a retention-recovery mechanism for cache magnitudes and unmergable tokens. However, such operations are not directly compatible in mainstream open-source KV quantization frameworks. Furthermore, its use of the SLERP function imposes several constraints, making it inapplicable to quantized caches, which fundamentally differs from XQuant.

H Configurations

The Configurations of XQuant in our main experiments are summarized in Table 10

I XQuant Pseudo Code

The pseudo code for the whole workflow is provided in Algorithm 1 and 2.

Model	Dataset	kq	vq	km	vm	eta1	eta2
Mistral-7b-v0.3	TruthfulQA	30	2	32	16	0	0
	HQA	30	2	32	16	1/6	0.045
	2Wiki	32	0	32	16	0	0.09
	MSQ	32	0	32	16	1/6	0
Mistral-7b-instruct-v0.2	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	TruthfulQA	28	0	32	28	1/3	0
	HQA	28	0	32	28	1/6	0.045
	2Wiki	28	0	32	28	1/3	0.045
	MSQ	28	0	32	28	1/3	0
Llama2-7b-chat	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 10: 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$\left(X_k^l,\ 2 \text{ if } l < kq \text{ else } 1\right)$
    else
        KeyCache$[l] \leftarrow$ PseudoQuantize$\left(X_k^l,\ 2 \text{ if } l < kq \text{ else } 1\right)$
    if $l < vq$ or $l \bmod 2 == 0$ then
        ValueCache$[l] \leftarrow$ Quantize$\left(X_v^l,\ 2 \text{ if } l < vq \text{ else } 1\right)$
    else
        ValueCache$[l] \leftarrow$ PseudoQuantize$\left(X_v^l,\ 2 \text{ if } l < vq \text{ else } 1\right)$
for $l \leftarrow 0$ to 31 do
    if $l < km$ or $l \bmod 2 == 0$ then
        DequantizedKey $\leftarrow$ Dequantize(
            KeyCache$[l][0]$,
            KeyCache$[l][1]$,
            KeyCache$[l][2]$)
    else
        DequantizedKey $\leftarrow$ Dequantize(
            KeyCache$[l-1][0]$,
            KeyCache$[l-1][1]$,
            KeyCache$[l][2]$)
    if $l < vm$ or $l \bmod 2 == 0$ then
        DequantizedValue $\leftarrow$ Dequantize(
            ValueCache$[l][0]$,
            ValueCache$[l][1]$,
            ValueCache$[l][2]$)
    else
        DequantizedValue $\leftarrow$ Dequantize(
            ValueCache$[l-1][0]$,
            ValueCache$[l-1][1]$,
            ValueCache$[l][2]$)
```

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 $\eta$;
    $scaling\_factor\_cali \leftarrow scaling\_factor \cdot (1 - 2 \cdot \eta[n\_bits])$ // Adjust scaling factor based on $\eta$;
    return
        $zero\_point\_cali$, $scaling\_factor\_cali$ // Return calibrated values;
```