LPRQuant: Low-Rank and Predictive KV Cache Residual Quantization for LLMs

Anonymous ACL submission

Abstract

The key-value (KV) cache in large language models is a critical memory bottleneck during inference. We propose LPRQuant , a training-free compression framework based on hierarchical residual approximation. LPRQuant decomposes KV cache compression into two stages: a structured first-level approximation followed by residual quantization, so that the joint approximation error is multiplicatively bounded and strictly smaller than either stage alone. Crucially, we treat the key and value caches asymmetrically based on their distinct structural properties: the key cache, which exhibits strong low-rank structure, is compressed via PCA projection plus residual quantization; the value cache is compressed via cross-cache linear prediction from the reconstructed keys, followed by residual quantization of the prediction residual. Experiments on LongBench and RULER across three model families (Llama, Mistral, Qwen) and scales (7B–32B) show that LPRQuant consistently outperforms existing methods, achieving near-lossless quality at ∼3-bit and retaining over 98% of full-precision performance at ∼2.5-bit.

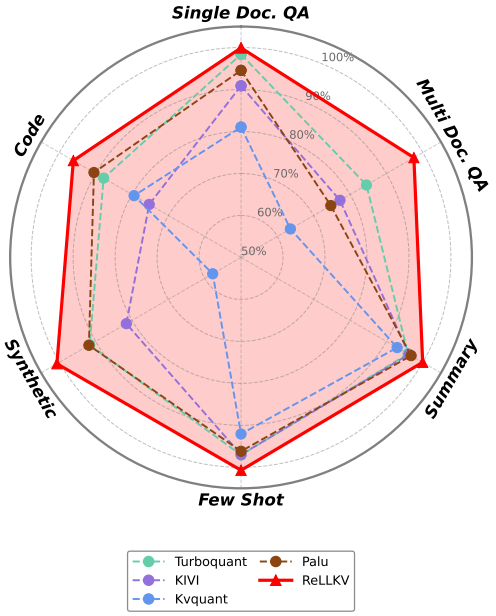

Figure 1: Relative performance (% of full cache) across six LongBench categories on Llama-3.1-8B-Instruct at the 2.5-bit setting. LPRQuant achieves the highest relative performance across all categories and most closely tracks the full-cache baseline among all methods.

1 Introduction

Large language models (LLMs) (Grattafiori et al., 2024; Qwen Team, 2024; Jiang et al., 2023) have demonstrated remarkable capabilities across a wide range of natural language processing tasks (Wei et al., 2022). However, deploying these models for long-context inference remains challenging due to the ever-growing memory footprint of the key-value (KV) cache. As context length scales, the KV cache can occupy a significant portion of GPU memory, severely limiting throughput and the deployable context length in practical deployments (Kwon et al., 2023).

To address this, existing work has explored two main directions for KV cache compression. The first line of work reduces the number of tokens stored in the cache, either by evicting less important tokens during generation (Li et al., 2024; Feng et al., 2025, 2026a,b) or by merging similar tokens (Zhang et al., 2024; Wang et al., 2024). While effective, these methods inevitably discard information that may be critical for future attention computation. The second direction reduces the *precision* of each cached token via quantization, retaining the full sequence at lower bit-widths; methods such as KIVI (Liu et al., 2024b), KVQuant (Hooper et al., 2024), and TurboQuant (Zandieh et al., 2025) have demonstrated that KV caches can be quantized to 4 bits with acceptable accuracy loss. However, existing quantization methods share a critical limitation: they treat the key and value caches as structurally homogeneous, applying the same class of compres-

sion without a principled understanding of their underlying differences. As we will show, this assumption is incorrect: the key and value caches exhibit fundamentally distinct structural properties that suggest very different, and more effective, compression strategies.

In this work, we systematically analyze the key and value caches across multiple LLMs and uncover two key structural observations. **First**, the key cache exhibits a pronounced low-rank structure: as shown in Figure 3, fewer than 10% of principal components suffice to capture over 90% of the variance in the key cache, suggesting that keys can be effectively approximated in a low-dimensional subspace. In contrast, the value cache exhibits a much weaker low-rank structure, requiring over 40% of the total rank to capture the same 90% of variance. **Second**, despite lacking low-rank structure, the value cache is strongly correlated with the key cache: across all layers, a simple linear map from key to value representations achieves a relative prediction error well below 1.0 (ranging from 0.29 to 0.58, Figure 5), confirming that the key representations are predictive of their corresponding value representations. Together, these observations reveal that the key and value caches are structurally distinct, motivating an *asymmetric* compression design.

Specifically, LPRQuant adopts an asymmetric compression strategy tailored to the distinct structural properties of each cache. For the key cache, we exploit its low-rank structure via PCA, compressing the majority of the variance into a compact subspace. For the value cache, we leverage its correlation with the key cache, learning a linear map to predict value representations from their corresponding key representations. In both cases, the residual is further quantized at low bit-widths, enabling aggressive compression while preserving the information critical for attention computation.

Contributions. Our contributions are as follows:
1 Structural analysis. We systematically analyze the structural differences between the key and value caches in LLMs, revealing that the key cache exhibits low-rank structure while the value cache exhibits strong correlation with the key cache, and demonstrate that these properties can be explicitly exploited for compression.
2 Method. We propose LPRQuant , a hierarchical KV cache compression framework that treats keys and values asymmetrically: low-rank projection combined with residual quantization for keys, and linear prediction combined with residual quantization for values.
3 Empirical validation. We demonstrate through extensive experiments across multiple models and benchmarks that LPRQuant consistently outperforms existing KV cache quantization methods at both ∼3-bit and ∼2.5-bit settings, achieving state-of-the-art performance across benchmarks (Figure 1).

2 Related Work

KV Cache Token Compression. A line of work reduces KV cache memory by retaining only a subset of tokens. Token eviction methods selectively discard tokens deemed less important based on attention scores (Li et al., 2024), activation magnitudes (Feng et al., 2025, 2026a), or learned policies (Feng et al., 2026b). StreamingLLM (Xiao et al., 2024) takes a different approach by retaining a fixed set of attention sink tokens alongside a sliding window of recent tokens to support streaming long-context inference. Token merging methods instead aggregate similar tokens into a compact representation to avoid hard eviction (Zhang et al., 2024; Wang et al., 2024). While these approaches can achieve high compression ratios, they inevitably discard or alter token-level information that may be relevant for future attention steps. In contrast, our method retains the full token sequence and instead compresses along the feature dimension, avoiding information loss due to token dropping.

KV Cache Quantization. Quantization-based methods reduce the memory footprint of KV cache by representing cached values at lower bit-widths. KIVI (Liu et al., 2024b) proposes to quantize the KV cache to 2 bits using per-channel quantization for keys and per-token quantization for values, while retaining recent tokens in full precision. KVQuant (Hooper et al., 2024) introduces per-channel and per-vector quantization strategies tailored to the distributional properties of KV cache. TurboQuant (Zandieh et al., 2025), which we adopt as our quantization backend, further improves quantization quality through rotation-based preprocessing. PaLu (Chang et al., 2025) decomposes the K/V projection weight matrices into low-rank matrix pairs via SVD, caching only the low-dimensional latent representations during inference, combined with quantization on the compressed representations. While some methods assign different bit-

widths to keys and values empirically (Liu et al., 2024b), or apply low-rank projection symmetrically to both caches (Chang et al., 2025), none explicitly exploits the distinct structural properties of keys and values in a principled asymmetric compression framework. Our work builds upon quantization as a second-level compression step, and demonstrates that explicitly accounting for the structural asymmetry between keys and values leads to substantially better compression quality.

3 Preliminaries

During autoregressive inference, transformer-based large language models cache the key and value states of each attention layer to avoid redundant computation. Formally, let $K \in \mathbb{R}^{B\times H\times S\times D}$ and $V \in \mathbb{R}^{B\times H\times S\times D}$ denote the key and value cache respectively, where B is the batch size, H is the number of attention heads, S is the sequence length, and D is the head dimension. As S grows, the memory footprint of the KV cache scales linearly, becoming a critical bottleneck in long-context inference. The goal of KV cache compression is to reduce the memory footprint of K and V while preserving the quality of the attention output. For convenience, we reshape K and V into $X \in \mathbb{R}^{N\times HD}$ where $N = B \times S$, concatenating all heads into a single feature dimension. Quantization reduces memory by representing floating-point values (e.g., 16-bit) with low-bit integers (e.g., 2–4 bits), at the cost of introducing approximation error. The lower the bit-width, the more aggressive the compression but the larger the quantization error.

4 Method

We propose LPRQuant (Figure 2), a hierarchical residual approximation framework that comprises two asymmetric compression paths: low-rank projection with residual quantization for the key cache, and linear prediction with residual quantization for the value cache. In both paths, residual quantization is performed using TurboQuant (Zandieh et al., 2025) as the quantization backend.

4.1 Hierarchical Residual Approximation

A naive approach to KV cache compression is to directly quantize K and V. However, the compression quality is fundamentally limited by the information loss introduced in a single approximation step. To address this, LPRQuant decomposes the compression into two complementary levels.

Framework. Given a target cache y, the first-level approximation $y^{(1)}$ captures the dominant structure of y. The remaining residual:

$$r^{(1)} = y - y^{(1)}$$

is then approximated by the second-level method, yielding $\tilde{r}^{(1)}$. The final reconstruction is:

$$\hat{y} = y^{(1)} + \tilde{r}^{(1)}$$

Error Analysis. We show that the joint reconstruction error satisfies:

$$\epsilon_{\text{joint}} = \epsilon_1 \cdot \epsilon_2$$

where $\|\cdot\|_F$ denotes the Frobenius norm, $\epsilon_1 = \|r^{(1)}\|_F/\|y\|_F$ and $\epsilon_2 = \|r^{(1)} - \tilde{r}^{(1)}\|_F/\|r^{(1)}\|_F$ are the relative errors of the first and second levels respectively. Since $\epsilon_1, \epsilon_2 \in (0, 1)$, the joint error is strictly smaller than either individual error, providing intuition for the hierarchical design over single-level methods. The detailed derivation is provided in Appendix A.

Application to KV Cache. The key insight of our framework is that although the two-level structure is unified, the choice of first-level approximation should be tailored to the intrinsic structure of each cache. As we will show in the following sections, K and V exhibit fundamentally different structural properties, which naturally leads to an asymmetric compression design:

- For K, which exhibits strong low-rank structure, we employ low-rank approximation as the first-level method (Section 4.2).
- For V, which lacks such low-rank structure, we exploit the inherent correlation between $\tilde{K}$ and V to construct a linear prediction as the first-level approximation (Section 4.3).

In both cases, quantization is applied to the residual as the second-level method.

4.2 K Cache Compression

Motivation. Prior to applying our hierarchical framework, we first examine the intrinsic structure of K. As shown in Figure 3, the cumulative eigenvalue energy of $X^\top X$ rises rapidly for K cache: fewer than 10% of dimensions already capture over 90% of the total energy. Since the low-rank approximation error satisfies

$$\|K - KPP^\top\|_F^2 = \sum_{i>r} \lambda_i(X^\top X),$$

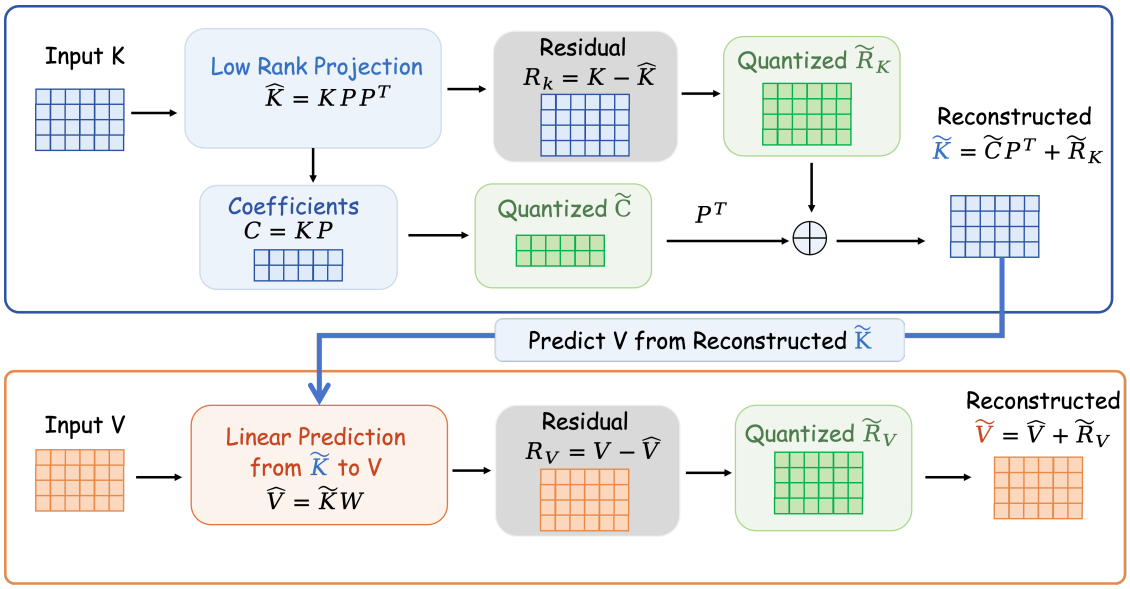

Figure 2: Overview of LPRQuant : asymmetric hierarchical residual approximation for K cache (low-rank projection + residual quantization) and V cache (linear prediction + residual quantization).

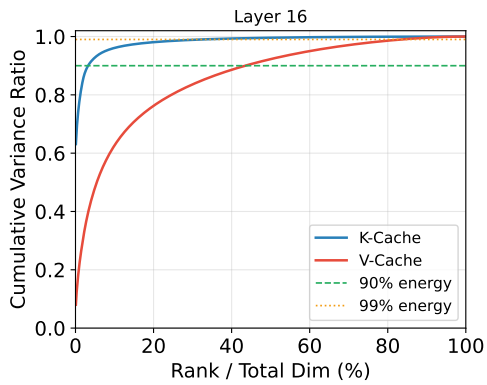

Figure 3: Cumulative eigenvalue energy ratio of $X^\top X$ for K cache and V cache at Layer 16 of Llama-3.1-8B-Instruct. K cache reaches 90% energy with fewer than 10% of dimensions, while V cache requires more.

where $P \in \mathbb{R}^{HD \times r}$ denotes the projection matrix (formally defined below), this rapid energy decay directly implies a small first-level error ϵ_1 (Wang et al., 2025; Li et al., 2025), making low-rank projection an effective first-level approximation.

Low-Rank Projection. Specifically, P consists of the top-r eigenvectors of $X^\top X$, where r is determined by the target bit budget, and is computed offline during a calibration phase shared across all heads. The first-level approximation of K is then:

$$\hat{K} = KPP^\top$$

where the low-rank coefficients $C = KP \in \mathbb{R}^{N \times r}$ capture the principal structure of K.

Hierarchical Compression. Following the framework in Section 4.1, the residual is:

$$R_K = K - \hat{K}$$

Both the coefficients C and the residual R_K are then quantized using TurboQuant (Zandieh et al., 2025), yielding quantized approximations $\tilde{C}$ and $\tilde{R}_K$ respectively; the specific bit-widths assigned to each component are described in Appendix B. The final reconstruction is:

$$\tilde{K} = \tilde{C}P^\top + \tilde{R}_K$$

4.3 V Cache Compression

Motivation. Unlike K, the cumulative eigenvalue energy of $X^\top X$ for V cache rises slowly, indicating that V does not exhibit strong low-rank structure. As shown in Figure 3, V cache requires substantially more dimensions than K cache to capture the same amount of energy. This is further confirmed in Figure 4 (bottom): low-rank compression fails to reach the Direct 4-bit baseline even at the highest rank tested, incurring a large first-level error ϵ_1 and making low-rank projection an ineffective first-level method for V.

Instead, we observe that K and V are both derived from the same hidden states, and thus exhibit strong cross-cache correlation. As shown in Figure 5, the reconstructed $\tilde{K}$ can predict V with substantially lower relative error than the trivial zero

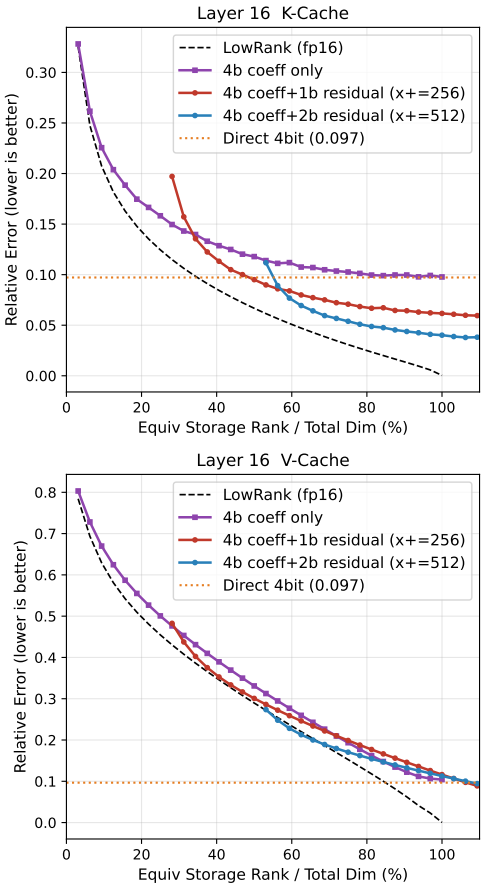

Figure 4: Relative reconstruction error of K cache (top) and V cache (bottom) at Layer 16 of Llama-3.1-8B-Instruct, under three configurations: 4-bit coefficients only, with 1-bit residual, and with 2-bit residual (ours). K cache hierarchical schemes consistently outperform direct 4-bit quantization, while V cache curves show no clear advantage across configurations.

baseline across all layers, validating the effectiveness of linear prediction as a first-level approximation for V. This motivates us to exploit $\tilde{K}$ as a predictor for V, constructing a linear prediction as the first-level approximation.

Linear Prediction. We fit a per-layer linear map $W \in \mathbb{R}^{HD \times HD}$ from $\tilde{K}$ to V via Ridge regression:

$$W = \arg\min_{W} \|\tilde{K}W - V\|_F^2 + \lambda \|W\|_F^2$$

where λ is the regularization strength. The closed-form solution is:

$$W = (\tilde{K}^\top \tilde{K} + \lambda I)^{-1} \tilde{K}^\top V$$

The matrix W is computed offline during a calibration phase using the reconstructed $\tilde{K}$, ensuring consistency between calibration and inference. The first-level approximation of V is then:

$$\hat{V} = \tilde{K}W$$

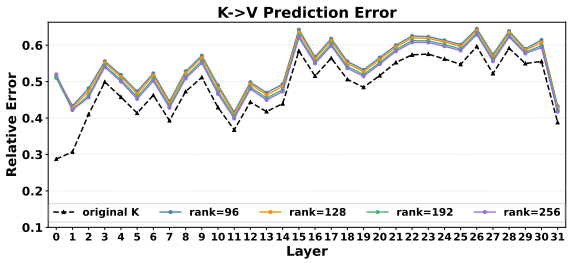

Figure 5: K→V linear prediction relative error across layers of Llama-3.1-8B-Instruct. Both the original K (dashed) and the reconstructed $\tilde{K}$ at different ranks (96, 128, 192, 256) predict V with relative error well below 1.0 across all layers, validating the effectiveness of exploiting cross-cache correlation as a first-level approximation for V.

Hierarchical Compression. Following the framework in Section 4.1, the residual is:

$$R_V = V - \hat{V}$$

The residual R_V is then quantized using TurboQuant (Zandieh et al., 2025), yielding $\tilde{R}_V$. The final reconstruction is:

$$\tilde{V} = \hat{V} + \tilde{R}_V$$

5 Experiments

5.1 Experimental Setup

Models. We evaluate LPRQuant on three open-source large language models: Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Mistral-7B-Instruct-v0.3 (Jiang et al., 2023), and Qwen2.5-32B-Instruct (Qwen Team, 2024), supporting context lengths of up to 128K, 32K, and 128K tokens respectively. Additional results on Qwen2.5-14B-Instruct are provided in Appendix D.

Baselines. We compare LPRQuant against four competitive KV cache compression methods: **TurboQuant** (Zandieh et al., 2025), **KIVI** (Liu et al., 2024b), **KVQuant** (Hooper et al., 2024), and **PaLu** (Chang et al., 2025). Detailed descriptions of each method are provided in Section 2. Method-specific implementation configurations are described in Appendix B.

Evaluation. We evaluate on LongBench (Bai et al., 2024), a comprehensive benchmark for long-context understanding, covering six categories: Single-Document QA (including NarrativeQA (Kočiskỳ et al., 2018), Qasper (Dasigi et al.,

Table 1: LongBench results across task categories on three models.

Method	Bits	Single-Doc QA	Multi-Doc QA	Summarization	Few-shot	Synthetic	Code	Avg.
Llama-3.1-8B-Instruct								
Full Cache	16	43.00	46.48	28.92	69.30	53.40	59.98	49.37
TurboQuant	3.0	42.20	40.15	28.29	68.33	**52.23**	54.56	46.91
KIVI	3.0	42.82	44.34	**29.04**	68.55	52.22	53.66	47.88
KVQuant	3.0	39.20	34.10	27.95	66.55	43.27	51.64	43.33
PaLu	3.0	41.44	38.32	28.57	67.44	51.50	56.63	46.47
LPRQuant	3.0	**43.67**	**44.94**	28.96	**69.49**	51.69	**61.08**	**49.17**
TurboQuant	2.5	42.30	39.28	27.75	67.26	48.94	52.66	45.81
KIVI	2.5	39.08	35.89	27.82	67.21	43.55	45.14	42.96
KVQuant	2.5	34.85	29.55	26.88	63.80	30.86	44.95	38.55
PaLu	2.5	40.66	34.72	27.99	66.65	49.05	54.26	44.79
LPRQuant	2.5	**42.97**	**45.34**	**28.91**	**69.79**	**53.75**	**57.66**	**48.99**
Mistral-7B-Instruct-v0.3								
Full Cache	16	38.38	38.80	28.90	70.81	51.50	63.26	47.51
TurboQuant	3.0	37.24	36.22	28.61	69.39	47.56	61.84	45.82
KIVI	3.0	38.23	**38.68**	28.56	70.39	48.34	**63.55**	46.96
KVQuant	3.0	35.61	30.91	27.44	68.07	39.92	59.61	42.82
PaLu	3.0	38.60	34.88	28.24	70.51	47.25	62.78	46.05
LPRQuant	3.0	**39.19**	38.52	**28.83**	**70.80**	**50.50**	63.10	**47.45**
TurboQuant	2.5	34.96	33.23	28.05	69.71	43.19	60.95	44.13
KIVI	2.5	33.47	33.08	28.07	68.00	45.37	59.69	43.62
KVQuant	2.5	30.80	23.73	27.48	66.95	30.62	57.39	38.93
PaLu	2.5	37.43	35.92	27.60	69.88	44.48	61.43	45.27
LPRQuant	2.5	**38.42**	**37.46**	**28.23**	**70.60**	**50.28**	**63.11**	**46.93**
Qwen2.5-32B-Instruct								
Full Cache	16	42.12	54.06	25.36	68.93	55.25	44.77	48.22
TurboQuant	3.0	41.37	51.36	24.32	69.98	52.40	**44.65**	47.20
KIVI	3.0	42.16	**53.96**	25.25	69.94	55.38	41.78	48.02
KVQuant	3.0	41.55	49.48	24.77	68.57	52.17	41.59	46.29
PaLu	3.0	40.83	53.35	25.05	69.49	54.38	30.80	46.03
LPRQuant	3.0	**42.71**	53.24	**25.54**	**70.23**	**56.25**	43.47	**48.41**
TurboQuant	2.5	39.82	47.84	23.91	67.88	51.65	38.44	44.91
KIVI	2.5	41.45	50.39	24.53	**69.54**	51.33	26.73	44.62
KVQuant	2.5	36.61	43.08	24.60	68.77	46.12	**40.53**	43.28
PaLu	2.5	41.41	52.43	25.13	69.37	52.29	23.27	44.76
LPRQuant	2.5	**42.27**	**53.41**	**25.67**	69.21	**53.09**	39.77	**47.34**

2021), MultiFieldQA-en), Multi-Document QA (including HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022)), Summarization (including GovReport (Huang et al., 2021), QMSum (Zhong et al., 2021), MultiNews (Fabbri et al., 2019)), Few-shot Learning (including TREC (Li and Roth, 2002), TriviaQA (Joshi et al., 2017), SAMSum (Gliwa et al., 2019)), Synthetic tasks, and Code (including LCC (Guo et al., 2023), RepoBench (Liu et al., 2024a)). Additionally, we evaluate on RULER (Hsieh et al., 2024) with a context length of 32K on Llama-3.1-8B-Instruct to assess long-context retrieval capability. Per-subtask breakdowns of LongBench results are provided in Appendix D.

Implementation Details. We calibrate the projection matrix P and the linear map W on 30 samples from RedPajama (Weber et al., 2024), each truncated to 4096 tokens. We evaluate two bit-budget settings: $\sim$3.0 bits and $\sim$2.5 bits per KV cache element. Detailed implementation settings are provided in Appendix B.

5.2 LongBench

Table 1 presents results across three models and two bit-budget settings. LPRQuant achieves the highest average score in all six configurations.

A notable trend is that the advantage of LPRQuant over baselines grows substantially as the bit budget decreases. For example, on Llama-3.1-8B-Instruct, the margin over the strongest base-

Table 2: Detailed scores of 13 subtasks on RULER (Llama-3.1-8B-Instruct, 32K context).

Method	cwe	fwe	niah_mk1	niah_mk2	niah_mk3	niah_mq	niah_mv	niah_s1	niah_s2	niah_s3	qa_1	qa_2	vt	Avg.
3-bit Setting														
Full Cache	31.34	87.53	99.6	99.6	99.4	99.00	99.4	100	100	100	79.4	55.8	99.36	88.49
TurboQuant	20.84	84.07	98.8	91.4	72.4	97.55	95.4	100	100	97.2	72.8	46.6	95.36	82.50
KIVI	29.28	88.80	99.2	96.6	95.2	98.80	97.1	100	99.6	99.6	75.2	51.0	96.64	86.70
KVQuant	16.80	80.47	73.8	57.8	11.6	77.55	77.75	99.4	87.0	72.4	54.0	38.2	75.08	63.22
PaLu	7.74	84.00	98.6	93.0	78.0	98.70	98.0	100	100	99.6	78.8	52.2	96.32	83.46
LPRQuant	45.56	88.87	99.2	99.6	99.6	99.30	97.5	100	100	100	78.4	51.2	99.12	89.10
2.5-bit Setting														
TurboQuant	11.14	63.93	97.4	81.6	44.6	97.50	92.7	100	99.4	97.2	66.8	44.0	90.28	75.89
KIVI	9.60	63.53	96.4	79.2	41.6	95.90	95.5	98.6	99.4	93.8	61.0	44.2	76.48	73.52
KVQuant	9.62	62.07	42.8	24.6	1.0	49.35	49.95	92.2	53.4	31.0	38.8	32.2	50.00	41.31
PaLu	37.52	81.40	99.8	90.4	72.4	97.80	95.15	100	100	96.8	69.6	49.6	89.76	83.09
LPRQuant	20.90	86.73	98.8	99.8	96.0	99.40	98.35	100	100	100	76.6	52.6	97.68	86.68

line widens from +1.29 at 3.0-bit to +3.18 at 2.5-bit. The same trend holds on Mistral-7B and Qwen2.5-32B, which represent different model families and scales, confirming that this advantage generalizes beyond a single architecture. This is consistent with our design motivation: as quantization becomes coarser, the residual after the first-level approximation remains small enough for low-bit quantization to handle, whereas direct quantization methods suffer from rapidly growing rounding errors.

At the 3.0-bit setting, LPRQuant nearly matches full-precision performance across all models: 49.17 vs. 49.37 on Llama (-0.4%), 47.45 vs. 47.51 on Mistral (-0.1%), and 48.41 vs. 48.22 on Qwen ($+0.4\%$). In particular, on Qwen2.5-32B, LPRQuant *exceeds* the full-precision baseline, suggesting that the structured projection may act as a beneficial regularizer for certain model.

The improvements are not uniform across task categories (Figure 1). Code completion and Synthetic tasks show the largest gains (e.g., on Llama at 2.5-bit, LPRQuant outperforms the next-best baseline by +3.40 on Code and +4.70 on Synthetic), whereas Summarization exhibits the smallest differences ($<$1 point across all methods). We attribute this to the differing attention patterns: code and synthetic retrieval tasks rely on precise long-range attention to specific tokens, which is more sensitive to KV cache distortion; summarization aggregates information diffusely across the context, making it more robust to quantization. Meanwhile, baseline methods show uneven degradation across categories: PaLu collapses on Code for Qwen2.5-32B (30.80 at 3.0-bit, 23.27 at 2.5-bit), and KVQuant degrades severely on Synthetic across all models at 2.5-bit. In contrast, LPRQuant reduces quantization error through first-level approximation before quantization, yielding more consistent performance across categories.

5.3 RULER

Table 2 presents the RULER results on Llama-3.1-8B-Instruct at 32K context length. Under the 3-bit setting, LPRQuant achieves an average of 89.10, slightly exceeding the full-precision baseline (88.49) and outperforming all compression methods. The advantage is most pronounced on the CWE (common words extraction) task, where LPRQuant scores 45.56 compared to 31.34 for full-precision and $\leq$29.28 for all baselines, suggesting that the hierarchical residual structure better preserves the fine-grained token-level information required by this task.

Under the more aggressive 2.5-bit setting, LPRQuant (86.68) maintains a clear lead over TurboQuant (75.89) and KIVI (73.52), with the NIAH multi-key subtasks showing the largest gaps. KVQuant degrades catastrophically to 41.31, consistent with the pattern observed on LongBench. Notably, PaLu performs competitively at 2.5-bit (83.09) due to its low-rank design, yet LPRQuant still outperforms it by +3.59 through the additional residual quantization stage.

5.4 Ablation Studies

We conduct ablation studies on Qwen2.5-32B-Instruct to validate the contribution of V cache linear prediction and analyze the sensitivity to the

Table 3: Component ablation on Qwen2.5-32B-Instruct (LongBench). TurboQuant: direct quantization of both K and V; +K Low+Res.: adds K low-rank projection with residual quantization; +V Pred.+Res.: further adds V linear prediction with residual quantization (full LPRQuant).

Configuration	Single-Doc QA	Multi-Doc QA	Summarization	Few-shot	Synthetic	Code	Avg.
Full Cache	42.12	54.06	25.36	68.93	55.25	44.77	48.22
3-bit Setting							
TurboQuant	41.37	51.36	24.32	69.98	52.40	44.65	47.20
+K Low+Res.	41.49	53.22	25.13	69.61	55.75	43.03	47.87
+V Pred.+Res.	42.71	53.24	25.54	70.23	56.25	43.47	**48.41**
2-bit Setting							
TurboQuant	39.82	47.84	23.91	67.88	51.65	38.44	44.91
+K Low+Res.	41.66	51.17	24.40	69.26	54.60	41.42	46.97
+V Pred.+Res.	42.16	54.00	25.44	70.17	55.25	41.36	**48.03**

Table 4: Effect of low-rank dimension r on Qwen2.5-32B-Instruct (LongBench, full LPRQuant pipeline).

Rank r	Single-Doc QA	Multi-Doc QA	Summarization	Few-shot	Synthetic	Code	Avg.
3-bit Setting							
96	42.76	54.08	25.47	68.51	55.00	42.13	47.92
128	42.46	52.84	25.38	69.97	54.79	45.00	48.22
192	41.99	53.80	25.66	69.29	54.75	45.12	48.25
256	42.71	53.24	25.54	70.23	56.25	43.47	**48.41**
2-bit Setting							
96	42.71	54.11	26.04	69.63	54.75	37.25	47.59
128	41.21	51.22	26.13	69.67	53.82	39.70	46.98
192	42.46	51.22	26.09	69.68	53.85	43.05	47.63
256	42.16	54.00	25.44	70.17	55.25	41.36	**48.03**

low-rank dimension r.

Component analysis. Table 3 shows the progressive effect of each component in LPRQuant (K low-rank with $r = 256$). Starting from TurboQuant (direct quantization of both K and V), adding K low-rank projection with residual quantization ("+K Low+Res.") already improves the average from 47.20 to 47.87 at 3-bit, confirming that the low-rank structure in K effectively reduces quantization error. Further adding V cache linear prediction with residual quantization ("+V Pred.+Res.", full LPRQuant) brings an additional +0.54, reaching 48.41, close to the full-precision score of 48.22. The same trend holds at the 2-bit setting, where the cumulative gain over TurboQuant reaches +3.12. Both stages provide complementary gains, validating the hierarchical design.

Effect of rank r. Table 4 reports results under the full LPRQuant pipeline (V: Pred.+Res.) with varying rank r. Performance is robust to the choice of rank: the average score varies by less than 0.5 points at 3-bit (47.92–48.41) and about 1.0 point at 2-bit (46.98–48.03). Higher ranks tend to improve the Code category, where preserving fine-grained head-level structure matters more, while other categories remain largely unaffected.

6 Conclusion

We presented LPRQuant , a training-free KV cache compression framework built on hierarchical residual approximation. By decomposing the compression into two stages: a structured first-level approximation (low-rank projection for K, linear prediction for V) followed by residual quantization, LPRQuant ensures that the joint error is multiplicatively bounded and strictly smaller. The asymmetric treatment of K and V caches is motivated by their distinct spectral properties: K exhibits strong low-rank structure suitable for PCA projection, while V benefits from cross-cache linear prediction using the reconstructed K. Experiments on LongBench and RULER across multiple model families and scales demonstrate that LPRQuant consistently outperforms existing compression methods, with particularly large gains under aggressive bit budgets where baseline approaches degrade significantly, validating the hierarchical residual design.

References

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 *Proc. ACL*.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed Abdelfattah, and Kai-Chiang Wu. 2025. Palu: Kv-cache compression with low-rank projection. In *Proc. ICLR*.

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

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

Yuan Feng, Haoyu Guo, JunLin Lv, S Kevin Zhou, and Xike Xie. 2026a. Taming the fragility of kv cache eviction in llm inference. In *Proc. ICLR*.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. 2025. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. In *Proc. NeurIPS*.

Yuan Feng, Junlin Lv, Haoyu Guo, Yukun Cao, Xike Xie, and S Kevin Zhou. 2026b. Identify critical KV cache in LLM inference from an output perturbation perspective. In *Proc. ICLR*.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization (EMNLP)*.

Aaron Grattafiori, Abhimanyu Dubey, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

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

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

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? In *Proc. COLM*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proc. NAACL*.

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

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proc. ACL*.

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

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 *Proc. SOSP*.

Guangyan Li, Yongqiang Tang, and Wensheng Zhang. 2025. Mgaa: Multi-granular adaptive allocation for low-rank compression of llms. *Preprint*, arXiv:2507.03294.

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

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

Tianyang Liu, Canwen Xu, and Julian McAuley. 2024a. Repobench: Benchmarking repository-level code auto-completion systems. In *Proc. ICLR*, volume 2024.

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. In *Proc. ICML*.

Qwen Team. 2024. Qwen2.5: A party of foundation models. https://qwenlm.github.io/blog/qwen2.5/. Blog Post.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. 2025. Svd-llm: Truncation-aware singular value decomposition for large language model compression. In *Proc. ICLR*.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*.

Maurice Weber, Daniel Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. 2024. Redpajama: an open dataset for training large language models. In *Proc. NeurIPS*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, and 1 others. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *Proc. ICLR*.

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

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. 2025. Turboquant: Online vector quantization with near-optimal distortion rate. In *Proc. ICLR*.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. 2024. Cam: Cache merging for memory-efficient llms inference. In *Proc. ICML*.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. Qmsum: A new benchmark for query-based multi-domain meeting summarization. In *Proc. NAACL*.

A Analysis of Hierarchical Residual Approximation

We provide the detailed derivation of the joint reconstruction error for the hierarchical residual approximation framework.

Setup. Let y denote the target to be approximated, and let $y^{(1)}$ denote the first-level approximation. Define the first-level residual as:

$$r^{(1)} = y - y^{(1)}$$

and the first-level relative error as:

$$\epsilon_1 = \frac{\|r^{(1)}\|_F}{\|y\|_F}$$

Second-Level Approximation. Rather than approximating y directly, the second-level method approximates the residual $r^{(1)}$. Let $\tilde{r}^{(1)}$ denote this approximation. The final reconstruction is then:

$$\hat{y} = y^{(1)} + \tilde{r}^{(1)}$$

Error Derivation. The final reconstruction error is:

$$\begin{aligned} y - \hat{y} &= y - \left(y^{(1)} + \tilde{r}^{(1)}\right) \\ &= \left(y - y^{(1)}\right) - \tilde{r}^{(1)} \\ &= r^{(1)} - \tilde{r}^{(1)} \end{aligned}$$

This is a key observation: the final error is not the approximation error with respect to the original target y, but rather the remaining error in approximating the first-level residual $r^{(1)}$.

Define the relative error of the second-level approximation on the residual as:

$$\epsilon_2 = \frac{\|r^{(1)} - \tilde{r}^{(1)}\|_F}{\|r^{(1)}\|_F}$$

By definition:

$$\begin{aligned} \|y - \hat{y}\|_F &= \|r^{(1)} - \tilde{r}^{(1)}\|_F \\ &= \epsilon_2 \cdot \|r^{(1)}\|_F \\ &= \epsilon_2 \cdot \epsilon_1 \cdot \|y\|_F \end{aligned}$$

Dividing both sides by $\|y\|_F$, the final relative error is:

$$\epsilon_{\text{joint}} = \frac{\|y - \hat{y}\|_F}{\|y\|_F} = \epsilon_1 \cdot \epsilon_2$$

Implication. Since $\epsilon_1 < 1$ holds whenever the first-level approximation captures any signal, and $\epsilon_2 < 1$ holds whenever quantizing the residual is better than discarding it (both naturally satisfied by any reasonable method), we have $\epsilon_1, \epsilon_2 \in (0, 1)$ and thus:

$$\epsilon_{\text{joint}} = \epsilon_1 \cdot \epsilon_2 < \min(\epsilon_1, \epsilon_2)$$

This demonstrates that hierarchical approximation achieves a joint error strictly smaller than either individual error alone. Furthermore, the better the first-level approximation (i.e., the smaller ϵ_1), the greater the reduction in joint error, which motivates the design of a strong first-level approximation tailored to the intrinsic structure of each target.

B Implementation Details

Baseline configurations. For all baselines, we apply the following method-specific configurations. **TurboQuant**: we adopt the MSE-optimized variant, which we empirically find to consistently outperform the information-theoretic variant. **KIVI**: we use a group size of 128 (vs. 32 in the original paper), which yields better performance at the same bit budget, and retain the 128 most recent tokens in full precision, following the residual cache design of the original work. **KVQuant**: we adopt the dense-and-sparse configuration with 1% of numerical outliers preserved in full precision. **PaLu**: we replace the original per-token integer quantization with TurboQuant for the low-dimensional latent representations, as it yields superior quantization accuracy on the compressed representations. The projection matrices are calibrated on the same 30 RedPajama samples (4096 tokens each) as LPRQuant to ensure a fair comparison.

Bit allocation. At the ∼3-bit setting, all methods uniformly allocate 3 bits to both the K and V caches (K3V3). For LPRQuant at this setting, the K cache uses rank $r = 256$ with coefficients C quantized at 4-bit and residual R_K at 2-bit, and the V cache residual is quantized at 3-bit (except Llama-3.1-8B-Instruct, where $r = 192$). At the ∼2.5-bit setting, the allocation differs across methods due to their design constraints. TurboQuant, KIVI, and KVQuant only support integer bit-widths, so we use K3V2. PaLu achieves a non-integer effective bit-rate by adjusting the low-rank projection dimension, and we configure it for a uniform ∼2.5-bit allocation across both caches. For LPRQuant , the K cache is compressed with rank $r = 128$, with the low-rank

coefficients C quantized at 4-bit and the residual R_K quantized at 2-bit, yielding an effective rate of approximately 2.5 bits per element. For the V cache, we rank all layers by their K→V prediction error and assign 3-bit residual quantization to the half with the largest errors and 2-bit to the remaining half, achieving an average of ∼2.5 bits across layers.

Hyperparameters and hardware. The Ridge regression regularization coefficient is set to $\lambda = 0.1$. The linear mapping W is robust to this choice: the K→V relative prediction error remains within 0.484 ± 0.001 across $\lambda \in [0.0001, 1]$. All experiments are conducted on NVIDIA H20 GPUs.

C Overhead Analysis

Calibration. W is obtained offline via a one-time calibration process. We select 30 sequences of 4096 tokens, as the prediction error drops significantly from 5 to 30 sequences (a reduction of 0.023 in relative error) but shows diminishing returns beyond this point, with less than 0.005 improvement when increasing from 30 to 50 sequences.

Storage. Per layer, W and P require 2 MB and 0.5 MB of storage respectively (16-bit, at the maximum rank $r = 256$), totaling 2.5 MB, equivalent to approximately 2,560 tokens of 4-bit KV cache. As context length grows, this fixed overhead becomes increasingly negligible: at 32K it represents 7.8% of the per-layer KV cache, dropping to 2% at 128K.

Inference. LLM decoding operates in the memory-bound regime, where throughput is limited by memory bandwidth rather than compute. Our method introduces one additional matrix multiplication $\tilde{k}_i W$ per token per layer, but in return reduces memory traffic: instead of loading the full V cache, only the lower-bit residual R_V is fetched from memory. Under the roofline model, this trade of compute for reduced memory access is acceptable, as the bottleneck lies in data movement rather

Table 5: Detailed per-subtask LongBench scores for Llama-3.1-8B-Instruct, Mistral-7B-Instruct-v0.3, and Qwen2.5-32B-Instruct.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	Hotpot	2WikiQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PR-en	Lcc	RB-P	
Llama-3.1-8B-Instruct																	
Full Cache (16-bit)	29.00	44.42	55.58	57.51	48.58	33.35	34.45	25.67	26.64	73.00	91.61	43.29	6.79	100.00	65.05	54.92	49.37
3.0-bit																	
TurboQuant	27.19	44.92	54.50	52.71	38.32	29.42	33.60	25.09	26.18	71.00	90.98	43.01	5.46	99.00	58.04	51.09	46.91
KIVI	30.55	44.12	53.80	54.93	47.40	30.69	34.37	25.66	27.10	70.50	91.49	43.66	5.26	99.17	58.54	48.77	47.88
KVQuant	25.25	41.32	51.04	49.00	30.25	23.04	32.99	24.53	26.33	70.00	88.19	41.47	3.54	83.00	56.97	46.31	43.33
PaLu	28.75	43.82	51.74	50.79	39.66	24.52	33.86	26.12	25.72	66.50	91.40	44.43	5.00	98.00	58.05	55.22	46.47
LPRQuant	30.80	44.97	55.23	56.07	47.96	30.80	34.72	25.48	26.67	73.00	91.73	43.74	3.38	100.00	63.74	58.42	49.17
2.5-bit																	
TurboQuant	28.52	45.95	52.44	51.47	39.67	26.70	31.86	25.05	26.33	70.50	89.99	41.30	2.96	94.92	55.07	50.25	45.81
KIVI	25.59	42.95	48.71	46.90	34.95	25.83	32.94	24.19	26.33	69.00	88.98	43.64	1.43	85.67	44.75	45.53	42.96
KVQuant	21.75	36.61	46.20	37.31	33.72	17.61	30.01	24.65	25.97	63.50	85.31	42.59	2.95	58.77	46.15	43.75	38.55
PaLu	26.53	43.42	52.03	48.15	33.90	22.11	33.21	25.56	25.21	66.50	88.74	44.71	3.00	95.12	56.49	52.02	44.79
LPRQuant	30.01	45.21	53.69	57.35	49.67	29.01	34.65	25.57	26.52	73.50	92.04	43.82	7.50	100.00	58.44	56.88	48.99
Mistral-7B-Instruct-v0.3																	
Full Cache (16-bit)	27.07	37.91	50.16	50.80	37.68	27.92	34.58	25.48	26.65	76.50	88.96	46.96	6.50	96.50	66.09	60.42	47.51
3.0-bit																	
TurboQuant	25.86	36.83	49.04	49.05	35.77	23.84	33.58	25.41	26.84	74.00	88.51	45.65	7.12	88.00	64.30	59.39	45.82
KIVI	25.79	38.10	50.80	50.86	40.54	24.63	34.22	24.88	26.59	76.50	87.88	46.79	5.67	91.00	64.77	62.33	46.96
KVQuant	21.21	37.61	48.02	45.22	28.03	19.49	32.13	23.96	26.23	72.00	87.64	44.58	3.35	76.50	63.47	55.74	42.82
PaLu	25.61	39.89	50.31	45.19	34.41	25.03	33.13	25.19	26.39	76.00	88.99	46.54	4.50	90.00	63.83	61.74	46.05
LPRQuant	27.31	38.84	51.42	49.26	38.68	27.62	33.96	25.53	26.99	76.50	89.14	46.76	5.50	95.50	64.35	61.86	47.45
2.5-bit																	
TurboQuant	24.16	35.44	45.27	49.46	30.34	19.88	32.37	25.04	26.74	75.00	88.79	45.34	3.37	83.00	64.13	57.78	44.13
KIVI	23.03	32.04	45.35	42.69	35.25	21.30	33.21	24.50	26.49	73.50	86.23	44.26	5.24	85.50	61.99	57.39	43.62
KVQuant	15.74	31.91	44.76	34.56	21.15	15.48	31.25	24.66	26.53	72.50	84.75	43.61	2.50	58.75	61.97	52.80	38.93
PaLu	24.87	37.09	50.34	44.96	34.59	28.22	32.39	24.62	25.78	76.00	89.10	44.54	3.47	85.50	63.39	59.47	45.27
LPRQuant	27.43	36.67	51.16	49.16	36.00	27.23	32.94	24.97	26.78	76.00	89.93	45.88	5.56	95.00	66.05	60.18	46.93
Qwen2.5-32B-Instruct																	
Full Cache (16-bit)	30.15	45.08	51.13	63.38	60.67	38.12	30.19	23.09	22.79	73.50	87.58	45.70	10.50	100.00	51.34	38.20	48.21
3.0-bit																	
TurboQuant	30.25	44.96	48.90	62.77	57.63	33.67	28.43	22.19	22.34	74.50	87.95	47.49	6.62	98.17	52.73	36.57	47.20
KIVI	31.68	44.23	50.56	61.13	60.29	40.45	29.91	22.90	22.94	74.50	88.03	47.29	11.00	99.75	44.07	39.49	48.02
KVQuant	31.69	42.98	49.98	59.33	55.26	33.86	29.57	22.29	22.45	71.00	88.49	46.21	7.27	97.08	46.08	37.11	46.29
PaLu	30.24	41.76	50.50	62.34	58.74	38.98	29.53	23.29	22.33	73.00	88.75	46.72	8.75	100.00	34.45	27.16	46.03
LPRQuant	31.94	44.70	51.48	61.61	60.30	37.80	30.64	23.20	22.77	73.50	89.86	47.34	12.50	100.00	49.97	36.97	48.41
2.5-bit																	
TurboQuant	30.19	41.95	47.32	56.39	53.67	33.46	27.77	21.71	22.24	71.00	87.34	45.29	4.87	98.42	43.20	33.68	44.91
KIVI	30.36	43.05	50.93	60.60	58.44	32.12	28.11	23.09	22.40	74.00	89.03	45.60	5.07	97.58	25.79	27.67	44.62
KVQuant	24.73	37.84	47.26	54.01	46.92	28.30	29.06	22.15	22.58	71.50	88.39	46.43	5.50	86.75	45.40	35.66	43.28
PaLu	30.89	41.59	51.75	61.77	58.37	37.16	29.33	23.51	22.55	72.50	89.19	46.42	5.50	99.08	20.96	25.58	44.76
LPRQuant	31.36	43.94	51.51	63.33	59.74	37.16	30.28	23.39	23.35	72.50	89.01	46.13	7.00	99.17	45.07	34.48	47.34

Table 6: Detailed per-subtask LongBench scores for Qwen2.5-14B-Instruct.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	Hotpot	2WikiQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PR-en	Lcc	RB-P	
Full Cache (16-bit)	29.89	45.66	54.57	61.64	57.71	38.37	29.59	23.59	21.81	76.50	90.69	47.57	10.20	98.75	62.96	50.81	50.02
3.0-bit																	
TurboQuant	27.13	37.84	49.48	57.89	48.41	34.76	28.49	22.99	21.04	77.00	87.86	46.98	4.18	91.83	57.18	54.45	46.72
KIVI	30.89	44.55	53.41	59.81	55.98	36.28	29.01	22.28	21.78	77.00	89.10	47.15	5.76	92.58	63.52	52.27	48.84
KVQuant	27.02	41.12	47.80	54.85	42.42	25.00	28.45	22.47	21.55	74.50	88.51	45.91	1.87	71.75	57.02	55.08	44.08
PaLu	31.41	43.59	53.54	60.93	57.67	37.48	26.12	22.63	20.10	72.50	87.65	46.71	2.00	95.75	48.56	39.19	46.61
LPRQuant	28.44	43.97	51.93	62.75	54.91	35.23	30.50	23.44	22.54	77.50	88.53	47.64	15.28	97.25	59.33	49.52	49.30
2.5-bit																	
TurboQuant	28.36	34.85	48.13	56.01	50.23	31.71	27.25	22.23	20.64	75.50	87.55	45.65	4.71	84.42	55.38	56.07	45.54
KIVI	25.14	32.06	47.34	54.28	42.14	28.77	26.79	22.03	20.97	73.50	88.13	46.73	3.46	71.58	57.63	45.58	42.88
KVQuant	24.27	34.60	42.44	39.60	33.81	18.12	27.77	22.00	21.27	73.50	85.36	44.67	3.11	56.83	56.27	52.36	39.75
PaLu	28.94	44.77	49.79	60.42	55.30	39.68	27.81	23.20	21.07	72.50	90.93	45.15	2.50	94.00	42.29	36.15	45.91
LPRQuant	27.60	44.69	52.91	58.27	57.61	34.91	29.97	23.21	22.09	77.50	88.84	47.74	9.55	97.50	53.51	49.64	48.47

Table 7: Detailed per-subtask RULER scores for Mistral-7B-Instruct-v0.3 (32K context).

Method	**cwe**	**fwe**	**niah_mk1**	**niah_mk2**	**niah_mk3**	**niah_mq**	**niah_mv**	**niah_s1**	**niah_s2**	**niah_s3**	**qa_1**	**qa_2**	**vt**	**Avg.**
Full Cache (16-bit)	29.90	86.40	91.4	65.6	28.8	94.45	96.00	99.2	98.8	99.6	60.0	43.0	93.64	75.91
3.0-bit														
TurboQuant	18.54	89.00	71.4	40.0	1.6	79.05	90.50	95.0	96.8	92.0	49.0	38.0	83.28	64.94
KIVI	41.26	88.60	80.6	41.4	4.8	86.95	92.65	94.0	98.4	94.2	54.2	38.6	84.84	69.27
KVQuant	10.72	85.27	43.0	4.0	0.0	45.25	54.65	91.2	74.2	55.6	38.4	36.6	78.36	47.48
PaLu	16.58	84.93	74.4	29.8	5.6	85.90	88.95	99.8	99.2	96.4	52.8	37.6	77.20	65.32
LPRQuant	22.56	86.53	91.4	62.0	31.4	95.05	96.25	99.4	97.4	98.2	60.0	39.4	93.28	74.84
2.5-bit														
TurboQuant	20.60	88.40	55.8	23.0	1.0	62.75	85.40	99.6	96.2	85.6	47.0	34.8	82.40	60.20
KIVI	25.94	79.80	53.4	11.6	0.4	60.15	79.85	64.0	89.8	67.0	49.6	36.6	65.04	52.55
KVQuant	13.78	78.80	25.8	1.0	0.0	21.90	24.15	68.4	52.6	17.6	30.8	30.0	64.08	32.99
PaLu	19.54	86.13	72.6	19.4	1.8	82.85	88.50	100.0	99.2	98.6	47.2	36.6	65.40	62.91
LPRQuant	12.48	86.80	87.4	65.6	18.0	91.65	96.65	99.6	98.2	98.8	58.0	42.4	91.56	72.86

than arithmetic throughput.

D Additional Experimental Results

This appendix provides detailed per-subtask LongBench results for the three main models (Table 5) and for Qwen2.5-14B-Instruct (Table 6), per-subtask RULER results on Mistral-7B-Instruct-v0.3 (Table 7), and visualization of the structural analysis across model families (Figures 6–9).

D.1 LongBench Detailed Results

Table 6 presents per-subtask results on Qwen2.5-14B-Instruct. LPRQuant achieves 49.30 at 3-bit and 48.47 at 2.5-bit, retaining over 96% of full-cache performance (50.02) at the more aggressive setting while most baselines degrade more noticeably. The advantage is most pronounced on Synthetic tasks (PCount, PR-en), consistent with the pattern observed on the three main models.

D.2 RULER Results

Table 7 presents per-subtask RULER results on Mistral-7B at 32K context length. LPRQuant achieves 74.84 at 3-bit and 72.86 at 2.5-bit, closely matching the full-cache score of 75.91. The advantage is most pronounced on multi-key NIAH subtasks (niah_mk1–mk3), where baselines suffer substantial degradation. At 2.5-bit, LPRQuant retains 87.4 on niah_mk1 while all other methods drop below 73, demonstrating the robustness of the hierarchical residual design under aggressive compression.

D.3 Visualization

Figure 6 extends the cumulative eigenvalue energy ratio analysis of Figure 3 in the main text to all representative layers (Layer 0, 8, 16, 24, 31) on Llama-3.1-8B-Instruct, showing that the strong low-rank structure of K cache and the weak low-rank structure of V cache are consistent properties across all layers rather than layer-specific phenomena.

Figures 7–9 report the K→V linear prediction error per layer for Mistral-7B, Qwen2.5-14B, and Qwen2.5-32B, complementing the Llama result in Figure 5 of the main text. Across all models, the prediction error is stable across layers and robust

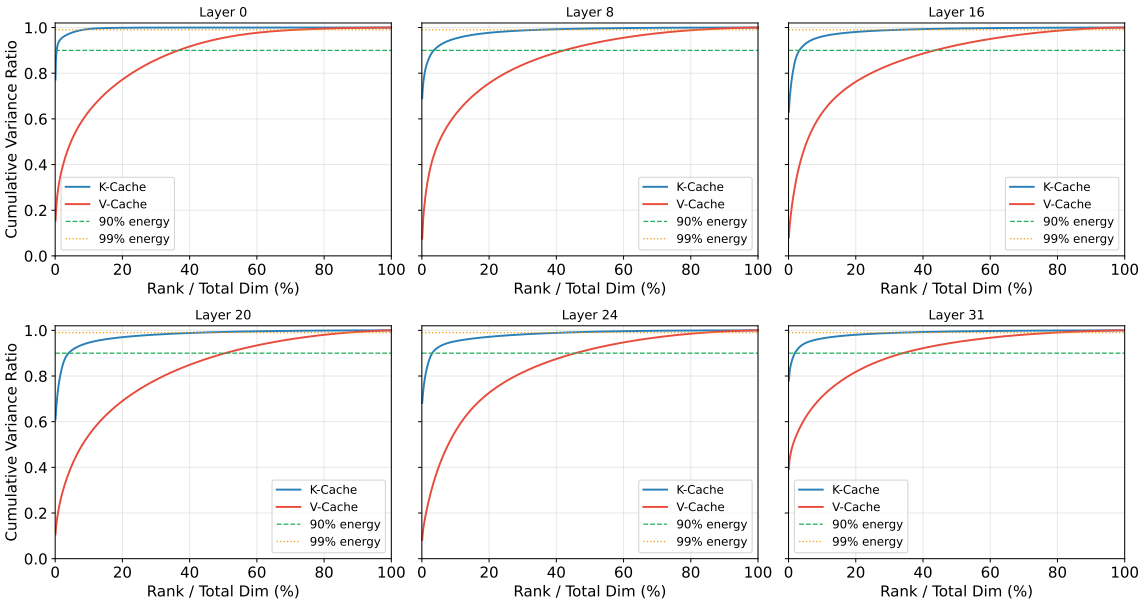

Figure 6: Cumulative eigenvalue energy ratio of $X^\top X$ for K cache and V cache across representative layers on Llama-3.1-8B-Instruct (Layer 0, 8, 16, 24, 31).

to rank choice ($r = 96$ to 256), consistent with the ablation results in Table 4.

Figures 10–13 show the low-rank compression error (complementing Figure 4 in the main text) across all representative layers on all four models. Across Llama, Mistral, Qwen2.5-14B, and Qwen2.5-32B, K cache consistently benefits from low-rank projection with residual quantization, while V cache shows no such advantage, confirming that the K/V structural asymmetry is a general property across model families and scales.

E Limitations and Future Work

Our method applies a uniform low-rank dimension r across all layers, and the K→V dependency is modeled as a linear mapping, which may not fully capture non-linear components in the key-value relationship. In future work, we plan to explore layer-adaptive rank allocation strategies and non-linear mapping functions to address these limitations. Furthermore, combining our approach with token eviction methods could provide complementary compression along both the hidden and sequence length dimensions, which we leave for future investigation.

F AI Assistance Disclosure

In the preparation of this manuscript, the authors utilized AI language models (Claude, Anthropic) to assist with language refinement and document formatting. Specifically, AI tools were employed to enhance the clarity and grammatical correctness of the writing, generate LaTeX code for tables and layout, and suggest structural organizations for the appendix. All core scientific contributions, including the conceptualization of the proposed method, algorithm design, experimental implementation, and the analysis of results, were conceived and executed entirely by the authors. The authors have carefully reviewed all AI-assisted content for accuracy and assume full responsibility for the final manuscript.

G Ethical Considerations

This work contributes to efficient LLM inference by reducing memory consumption and computational overhead, which in turn lowers the energy cost and carbon footprint of large-scale model deployment. By reducing hardware requirements, our method helps democratize access to advanced AI capabilities on resource-constrained devices. This work does not introduce any direct ethical concerns, as it focuses purely on inference efficiency optimization without modifying model behavior or training data.

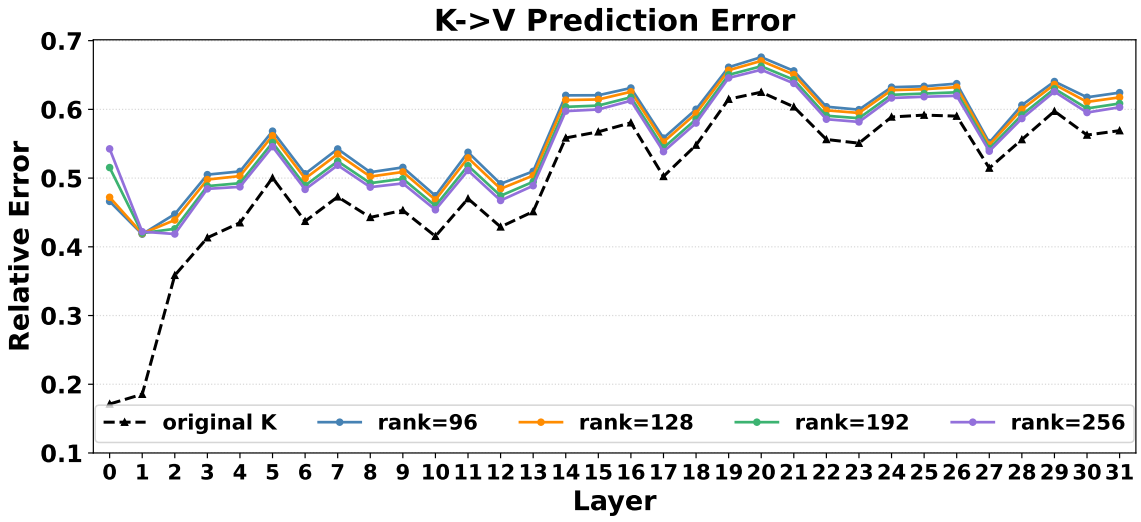

Figure 7: K→V prediction error per layer on Mistral-7B-Instruct-v0.3. Different rank choices yield comparable prediction quality across all layers.

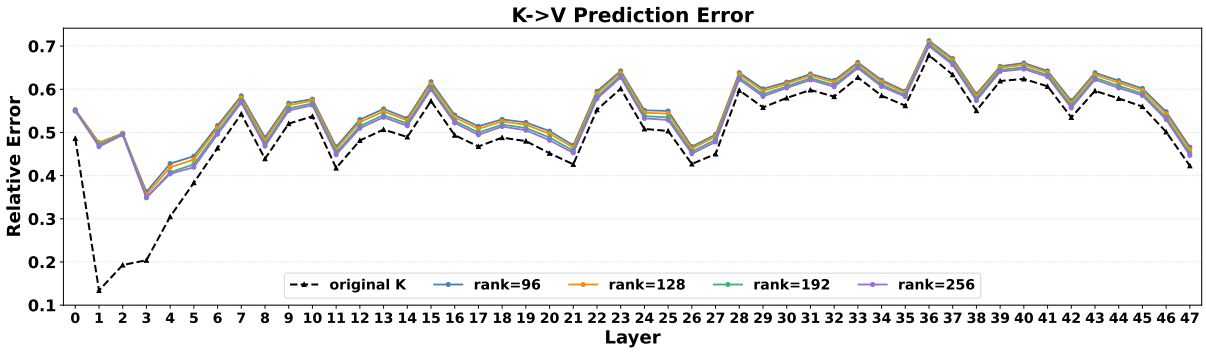

Figure 8: K→V prediction error per layer on Qwen2.5-14B-Instruct. The prediction error is stable across layers and robust to rank choice.

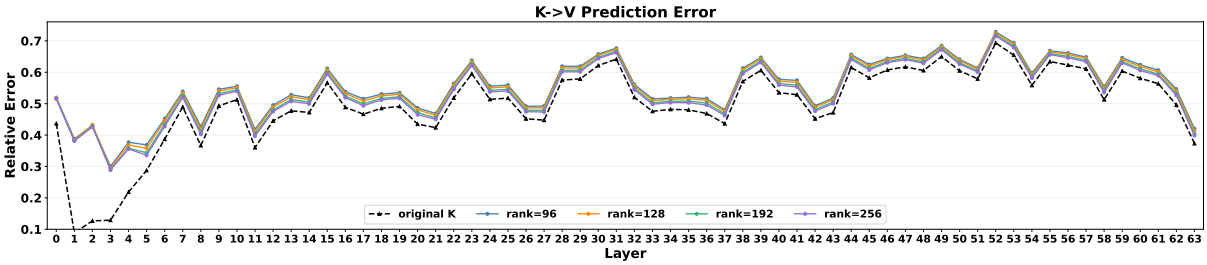

Figure 9: K→V prediction error per layer on Qwen2.5-32B-Instruct. The same pattern holds at the 32B scale with slightly lower overall error.

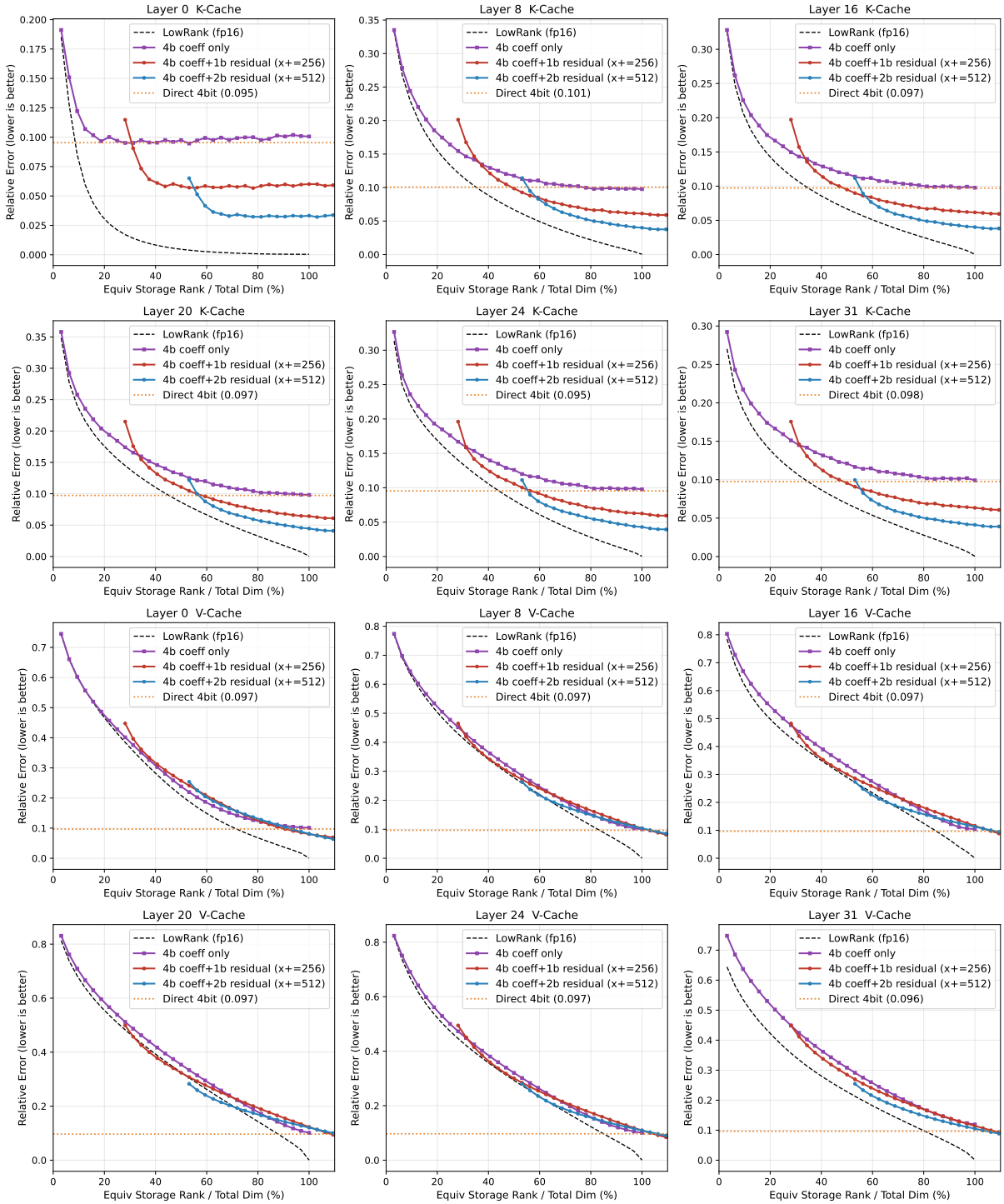

Figure 10: Low-rank compression error on Llama-3.1-8B-Instruct across representative layers (Layer 0, 8, 16, 24, 31). K cache benefits significantly from low-rank projection with residual quantization, while V cache shows no such advantage across all layers.

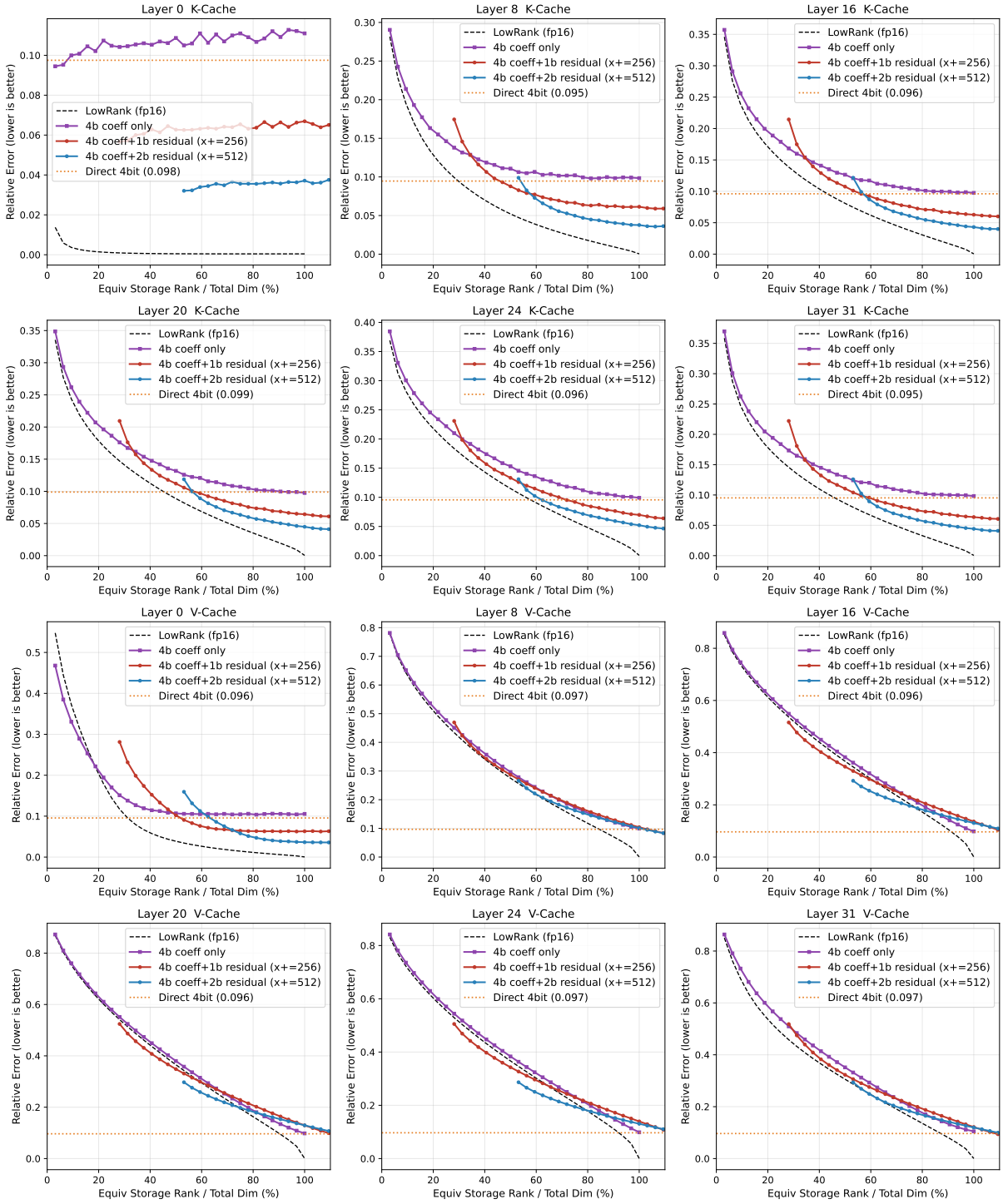

Figure 11: Low-rank compression error on Mistral-7B-Instruct-v0.3 across representative layers (Layer 0, 8, 16, 24, 31). The same K/V asymmetry observed on Llama holds on Mistral, confirming cross-family generality.

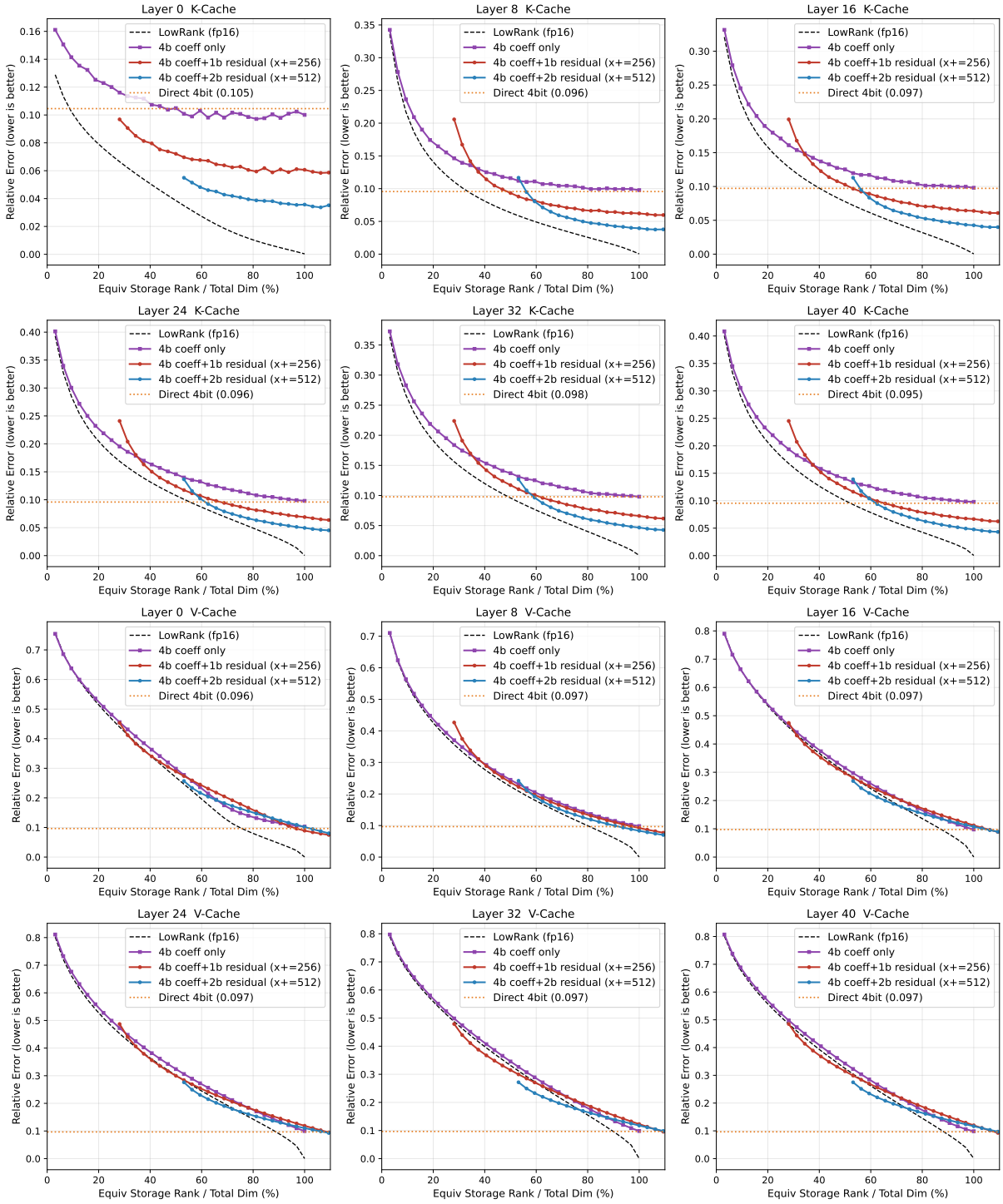

Figure 12: Low-rank compression error on Qwen2.5-14B-Instruct across representative layers. The K/V structural asymmetry persists at the 14B scale.

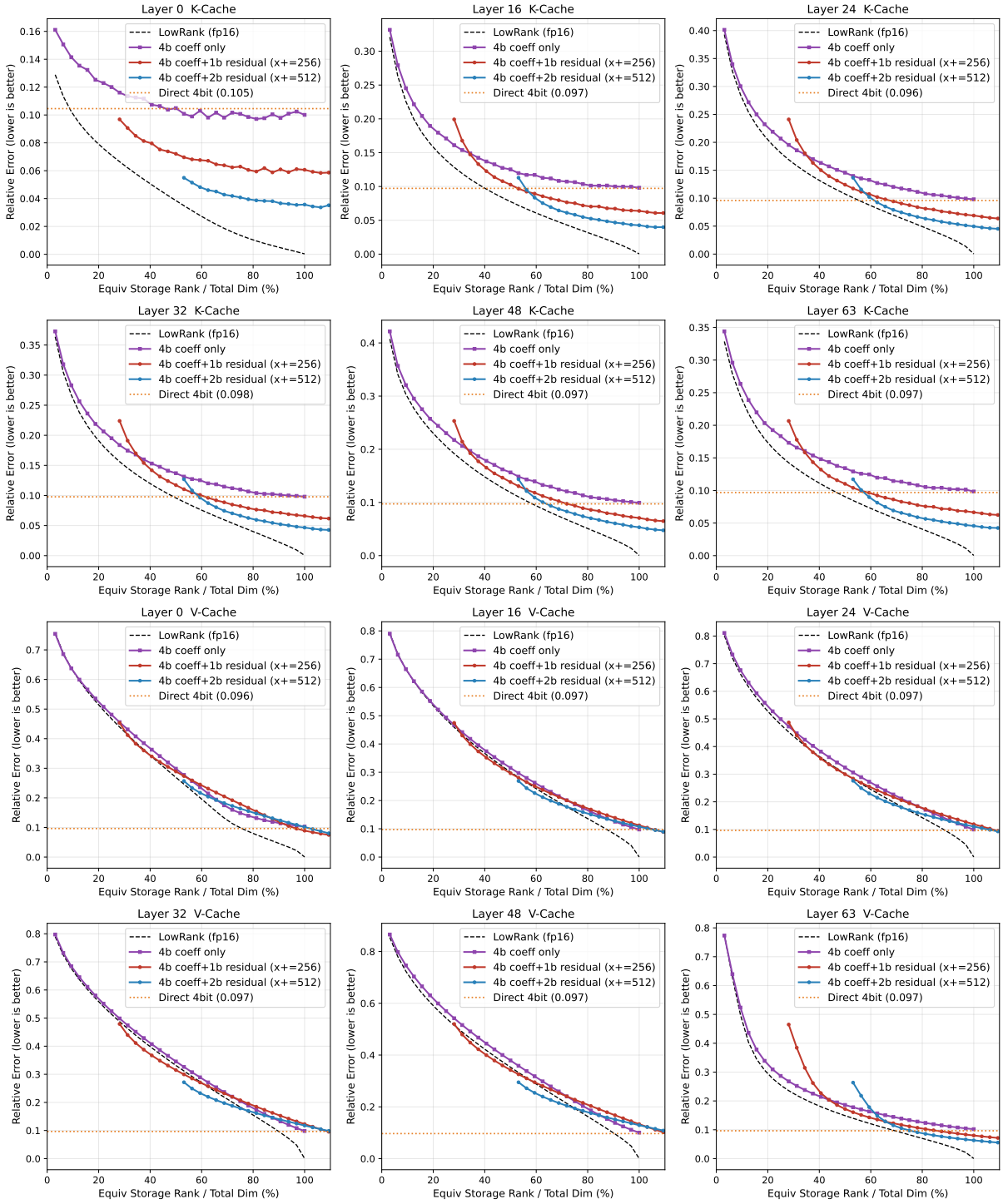

Figure 13: Low-rank compression error on Qwen2.5-32B-Instruct across representative layers. The asymmetry is consistent even at the 32B scale.