RankVQ: Low-Rank Parameterized Commutative Vector Quantization for KV Cache Compression

Jianglin Zhou[1], **Huaming Wu**[1,2]*, **Huijun Tang**[3]
[1]Center for Applied Mathematics, KL-AAGDM, Tianjin University, Tianjin 300072, China
[2]State Key Laboratory of Synthetic Biology, Tianjin University, Tianjin 300072, China
[3]School of Computer Science and Informatics, University of Liverpool, Liverpool L69 3DR, UK
{jianglin1207154288, whming}@tju.edu.cn, huijun.tang@liverpool.ac.uk

Abstract

The growing context lengths of large language models (LLMs) have made key-value (KV) cache memory a critical bottleneck during inference. Vector quantization (VQ) with commutative codebooks provides an effective Rotary Position Embedding (RoPE)-aware solution by enabling reconstruction-free attention over quantized key indices. However, existing commutative VQ methods still parameterize scalar codebook coefficients across RoPE subspaces and codewords largely independently, leaving parameter-side redundancy underexploited. We propose **RankVQ**, a low-rank parameterized commutative vector quantization method for KV cache compression. Instead of factorizing the deployed codebooks, RankVQ factorizes the underlying scalar coefficient matrices used to construct key and value codebooks and refines them through assignment-aware EM. This design preserves the pseudo-symmetric structure required for RoPE commutativity on the key side, while exploiting parameter-side redundancy during offline codebook construction. The low-rank factors are discarded after codebook construction, so RankVQ keeps the same online inference form as standard commutative VQ and introduces no additional decoding overhead. Experiments on LongBench, InfiniteBench, and GSM8K show that RankVQ achieves a stronger accuracy–compression trade-off than directly comparable KV quantization baselines, with particularly clear gains in the 1-bit regime.

1 Introduction

Large language models (LLMs) have demonstrated strong capabilities in long-document understanding [1], dialogue [2], code generation [3], and complex reasoning [4]. This progress has been accompanied by a rapid expansion of context windows. For instance, LLaMA 3.1 [5] supports contexts of up to 128K tokens, while recent long-context techniques [6, 7] aim to extend this limit even further. Despite these advances, efficient long-context inference remains challenging because the key-value (KV) cache grows linearly with sequence length. As a result, KV cache storage can dominate GPU memory consumption and become a primary bottleneck in deploying LLMs for long-context applications [8, 9].

Existing methods reduce the memory and computation overhead of KV caches mainly through token eviction, sparse attention, or quantization. Token eviction [10–12] and sparse attention [13–15] lower inference cost by discarding cached tokens or skipping part of the attention computation. However, these methods may hurt tasks that require faithful access to the full context. KV quantization [16–18] offers a complementary direction: it compresses cached representations while preserving the

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

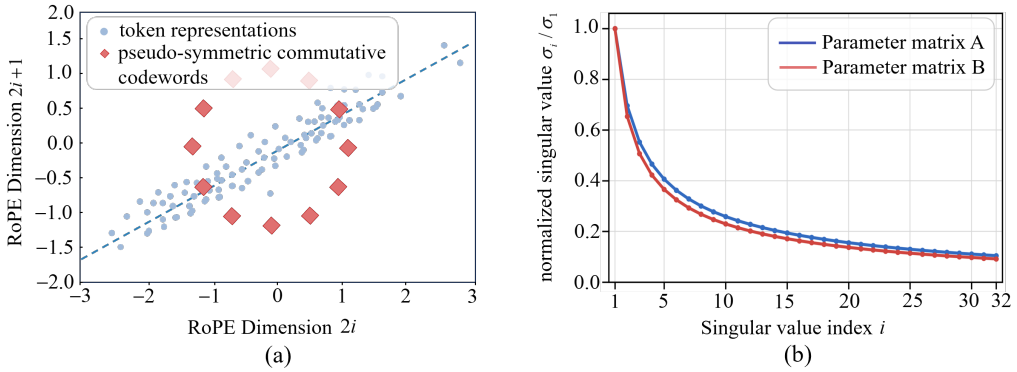

Figure 1: (a) Token representations in RoPE blocks exhibit pronounced anisotropic low-rank structure. (b) The normalized singular-value spectra of the learned parameter matrices A and B show clear parameter-side low-rank redundancy. Results are obtained from LLaMA-3.1-8B-Instruct under the default 1-bit codebook construction setting.

complete token sequence. Among quantization methods, vector quantization (VQ) [19–21] is especially attractive in ultra-low-bit settings, as it represents continuous KV vectors using compact discrete indices.

Within VQ-based KV cache compression, commutative codebooks [19] provide an effective mechanism for exploiting the algebraic structure of Rotary Position Embedding (RoPE) [22]. On the key side, each RoPE subspace is quantized by pseudo-symmetric codewords that commute with the corresponding RoPE rotation. This property allows attention scores to be computed directly from codeword indices without explicit key reconstruction, reducing decoding overhead during autoregressive inference. However, this design does not fully remove redundancy from the codebook parameters. As illustrated in Fig. 1(a), token representations within RoPE blocks are highly anisotropic and concentrate along a small number of dominant directions [23]. Consistently, Fig. 1(b) shows that the learned scalar parameter matrices A and B exhibit rapidly decaying singular-value spectra, indicating parameter-side low-rank redundancy. These findings motivate a structured codebook parameterization. Instead of treating codeword coefficients across subspaces and codewords as largely independent parameters, the codebook can exploit the shared low-dimensional structure observed in the learned coefficient matrices.

Motivated by these observations, we propose **RankVQ**, a low-rank parameterized commutative vector quantization method for KV cache compression. Instead of factorizing the deployed codebook, RankVQ factorizes the underlying scalar parameter matrices. This design preserves the pseudo-symmetric structure required for RoPE commutativity, while reducing parameter-side redundancy during codebook construction. The low-rank factors are initialized from a full-rank reference codebook using truncated SVD and are then refined through an Expectation-Maximization (EM) procedure that alternates between codeword assignment and low-rank factor updates [24]. Since these factors are used only for offline codebook construction and are discarded before deployment, RankVQ introduces no additional inference overhead compared with standard commutative codebooks. For clarity, we first present the single-stage formulation in the main text and then extend it to the grouped residual setting used in our experiments. The full details of the grouped residual extension are provided in Appendix B.4. We summarize our contributions as follows:

- **Parameter-side redundancy in RoPE-aware commutative VQ.** We identify and empirically analyze a form of redundancy that is largely overlooked in existing KV cache compression methods: the scalar coefficient matrices underlying commutative codebooks exhibit pronounced low-rank structure across RoPE subspaces and codewords. This finding motivates exploiting shared low-dimensional structure during offline codebook construction, rather than treating the codeword coefficients as independent parameters.
- **Low-rank parameterized commutative codebook construction.** We propose RankVQ, which factorizes the scalar coefficient matrices used to construct key and value codebooks and refines them through assignment-aware EM. Unlike post-hoc low-rank approximation of deployed codebooks, RankVQ preserves the pseudo-symmetric form required for RoPE

commutativity on the key side. The low-rank factors are used only during offline construction and are discarded before deployment.

- **Strong accuracy–compression and memory–latency trade-offs.** Experiments on long-context and reasoning benchmarks show that RankVQ improves the accuracy–compression trade-off over directly comparable KV quantization baselines, with especially clear gains in the 1-bit regime. Efficiency results further show that RankVQ reduces deployed KV cache memory while keeping the same online commutative attention form.

2 Related Work

Existing methods reduce KV cost through quantization, sparsity, or token eviction. Among VQ methods, CommVQ [19] uses pseudo-symmetric commutative codebooks to avoid explicit key reconstruction under RoPE. RankVQ keeps this inference form but introduces shared low-dimensional structure into the codebook coefficients during offline construction.

Low-rank factorization [25–27] has been used for parameter-efficient fine-tuning [28], weight compression [25], and activation compression [27]. Our setting is different: the low-rank structure is imposed on codebook parameters while preserving the pseudo-symmetric constraint on the key side, and the factors are discarded after offline construction, leaving deployed inference unchanged. More discussion is given in Appendix A.

3 Preliminaries

3.1 RoPE and the Commutativity Property

Rotary Position Embedding (RoPE) encodes token positions by applying a block-diagonal rotation to query and key vectors [22]. Since RankVQ exploits the commutative structure induced by RoPE on the key side, we focus on the rotated key representation. Throughout the paper, per-subspace queries and keys are represented as row vectors. Accordingly, RoPE is applied by right multiplication, which is transpose-equivalent to the standard column-vector formulation.

For a token at position m, let $k_m^i \in \mathbb{R}^{1\times 2}$ denote the i-th two-dimensional key sub-vector. RoPE rotates this sub-vector as

$$k_m^i \leftarrow k_m^i R_m^i, \qquad R_m^i = \begin{pmatrix} \cos(m\theta_i) & \sin(m\theta_i) \\ -\sin(m\theta_i) & \cos(m\theta_i) \end{pmatrix}, \tag{1}$$

where $\theta_i = 10000^{-2(i-1)/d}$ and d denotes the per-head hidden dimension. This block-wise rotation structure induces a commutativity property that is central to the construction of commutative codebooks.

Property 1 (Commutativity) *Let $C \in \mathbb{R}^{2\times 2}$ be a pseudo-symmetric matrix defined as*

$$C = aI + bJ = \begin{pmatrix} a & b \\ -b & a \end{pmatrix}, \quad a, b \in \mathbb{R} \tag{2}$$

where $a, b \in \mathbb{R}$ are scalar coefficients, $I \in \mathbb{R}^{2\times 2}$ is the identity matrix, and $J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$. Then $R_m^i C = C R_m^i$ for all token positions m and RoPE frequency indices i.

This property holds because both C and R_m^i belong to the same commutative matrix algebra $\{aI+bJ \mid a, b \in \mathbb{R}\}$.

3.2 Commutative Codebooks and Their Limitation

A commutative codebook quantizes each two-dimensional RoPE sub-vector $k_m^i \in \mathbb{R}^2$ using codewords that satisfy the pseudo-symmetric structure in Eq. 2. Let N_c denote the number of key side codewords per RoPE subspace, and let D denote the number of 2D RoPE subspaces, i.e., $D = d/2$. The l-th key side codeword in the i-th subspace is:

$$C_K^{i,l} = a_i^l I + b_i^l J, \qquad l \in \{0, \ldots, N_c - 1\}, \tag{3}$$

Stacking the scalar coefficients across all D subspaces and all N_c codewords yields two global parameter matrices:

$$A \in \mathbb{R}^{D \times N_c}, \qquad B \in \mathbb{R}^{D \times N_c}, \tag{4}$$

with entries $A_{i,l} = a_i^l$ and $B_{i,l} = b_i^l$. By Property 1, every codeword commutes with R_m^i, enabling reconstruction-free attention computation (see Section 4.4).

Limitation. Existing commutative codebooks parameterize codeword coefficients largely independently across RoPE subspaces and codewords, without exploiting shared low-dimensional structure. This parameterization can be redundant because the learned coefficient matrices exhibit rapidly decaying singular-value spectra, as shown in Fig. 1. These observations motivate RankVQ to impose a shared low-rank parameterization on the underlying codebook coefficients.

4 RankVQ

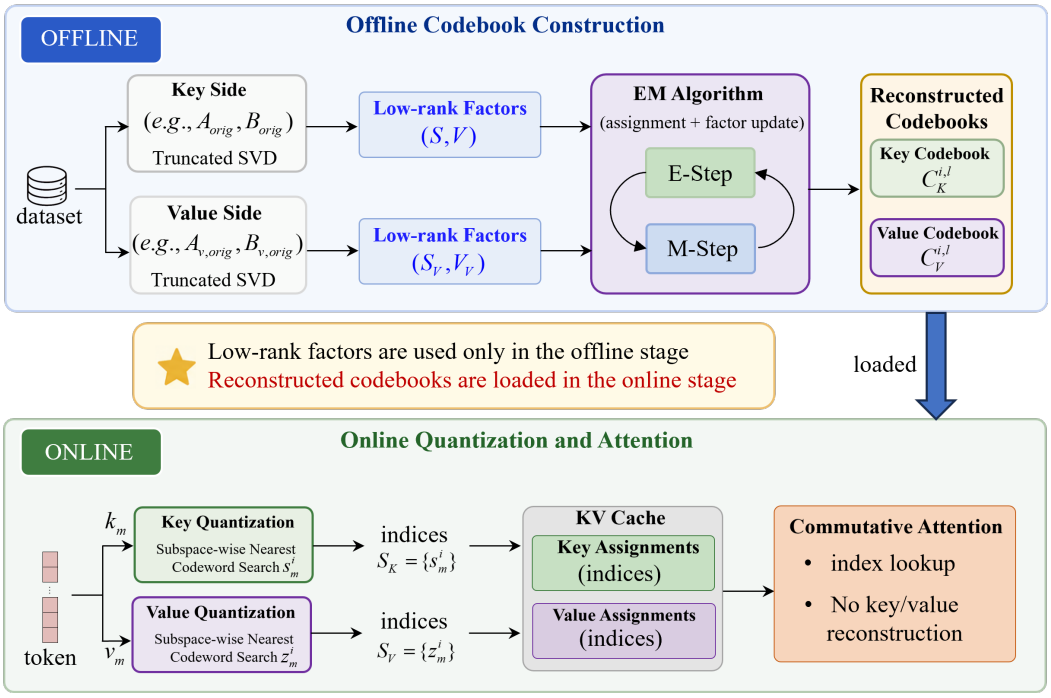

Figure 2: RankVQ Pipeline. Top: Offline low-rank parameterization via EM. Bottom: Online KV cache quantization with commutative attention.

4.1 Motivation: Why Low-Rank Parameterization?

Fig. 1 reveals two complementary forms of low-dimensional structure in commutative codebooks. On the data side, RoPE-block activations are highly anisotropic and concentrate along a small number of dominant directions. On the parameter side, the learned scalar coefficient matrices A and B exhibit rapidly decaying singular-value spectra, indicating substantial approximate low-rank redundancy. These observations suggest that independently parameterizing codeword coefficients across all RoPE subspaces and codewords is unnecessary. The two observations play different roles. The activation anisotropy provides the data-side intuition, while the singular-value spectra directly characterize the coefficient matrices that RankVQ factorizes. This spectral concentration persists across all residual stages for both key and value coefficient matrices (Appendix F.7).

RankVQ therefore introduces a shared low-dimensional parameter subspace for the scalar coefficients underlying commutative codebooks. The factorization makes different subspaces and codewords share a small set of coefficient modes. Moreover, for a fixed reconstructed coefficient matrix, the Frobenius regularization on the factors has the standard nuclear-norm variational form. For example,

$$\min_{S_A, V_A : S_A V_A^\top = A} \frac{1}{2} \left(\|S_A\|_F^2 + \|V_A\|_F^2 \right) = \|A\|_*. \tag{5}$$

It therefore favors coefficient matrices whose variation is concentrated in a small number of spectral directions. EM then updates the assignments together with this structured codebook, rather than approximating a fixed full-rank solution after training. At the default $N_r = 32 = \min(D, N_c)$, the hard rank constraint is inactive, while this regularization remains active. Importantly, this low-rank structure is imposed on the codebook parameters rather than on each deployed 2D pseudo-symmetric codeword. As a result, RankVQ preserves the commutativity required by RoPE while introducing a structured regularization over the codebook parameters during codebook construction. The same parameter-sharing principle can also be applied to value codebooks, providing a structured low-rank parameterization without requiring RoPE commutativity. Fig. 2 illustrates the overall RankVQ pipeline.

4.2 Low-Rank Parameterization

Key side. Let N_r denote the factor dimension. RankVQ parameterizes the key side scalar codebook matrices as

$$A = S_A V_A^\top, \qquad B = S_B V_B^\top, \tag{6}$$

where $S_A, S_B \in \mathbb{R}^{D \times N_r}$ are subspace-basis matrices and $V_A, V_B \in \mathbb{R}^{N_c \times N_r}$ are codeword-coordinate matrices. For the i-th RoPE subspace and the l-th codeword, we define the corresponding *vector-form* codeword as $c_K^{i,l} = \left[(S_A V_A^\top)_{i,l}, (S_B V_B^\top)_{i,l}\right] \in \mathbb{R}^{1\times 2}$. Its deployed *matrix-form* codeword is

$$C_K^{i,l} = (S_A V_A^\top)_{i,l} I + (S_B V_B^\top)_{i,l} J. \tag{7}$$

That is, if $c_K^{i,l} = [a_i^l, b_i^l]$, then $C_K^{i,l} = a_i^l I + b_i^l J$. Since the low-rank factorization is applied to the scalar coefficients rather than to the deployed 2×2 codeword matrix itself, every codeword remains pseudo-symmetric by construction. Therefore, the commutativity property in Property 1 still holds, and the inference-time attention computation retains the same algebraic form as in standard commutative-codebook inference.

Value side. The value side codebook is parameterized analogously. Let N_v denote the number of value side codewords per value sub-vector, and let D_v denote the number of value sub-vectors. We factorize the value side scalar matrices as $A_v = S_{A_v} V_{A_v}^\top$ and $B_v = S_{B_v} V_{B_v}^\top$, with $S_{A_v}, S_{B_v} \in \mathbb{R}^{D_v \times N_r}$ and $V_{A_v}, V_{B_v} \in \mathbb{R}^{N_v \times N_r}$. The deployed vector-form value codeword is

$$c_V^{i,l} = \left[(S_{A_v} V_{A_v}^\top)_{i,l}, (S_{B_v} V_{B_v}^\top)_{i,l}\right] \in \mathbb{R}^{1\times 2}. \tag{8}$$

For compact notation, we represent the two scalar value codeword coefficients analogously, although no commutativity constraint is imposed on the value side

$$C_V^{i,l} = (S_{A_v} V_{A_v}^\top)_{i,l}\, I + (S_{B_v} V_{B_v}^\top)_{i,l}\, J. \tag{9}$$

The value side does *not* require pseudo-symmetry or commutativity, since RoPE is applied only to keys. We nevertheless adopt the same low-rank parameterization on the value side because value codebook parameters exhibit similar parameter-side redundancy. Section 5.4 shows that applying low-rank EM refinement to the value-side parameters provides a substantial performance benefit.

This design differs from parameter-efficient fine-tuning methods such as LoRA [28] in purpose and deployment. In RankVQ, the low-rank factors are only used during offline codebook construction. After training, they are discarded and replaced by the reconstructed full codebooks.

4.3 Training with Low-Rank Parameterized EM

We train RankVQ on a calibration set of pre-RoPE key and value vectors. For clarity, we first present the single-stage case. The grouped residual deployment used in experiments is deferred to Appendix B.4. Training alternates between codeword assignment and low-rank factor updates on both the key and value codebooks.

Initialization. We first train a full-rank commutative codebook to obtain reference scalar parameter matrices $A_{\text{orig}}, B_{\text{orig}} \in \mathbb{R}^{D \times N_c}$ (key side) and $A_{v,\text{orig}}, B_{v,\text{orig}} \in \mathbb{R}^{D_v \times N_v}$ (value side). We initialize the low-rank factors via truncated SVD. For example, on the key side,

$$A_{\text{orig}} \approx S_A^{(0)} \Sigma_A^{(0)} (V_A^{(0)})^\top, \qquad B_{\text{orig}} \approx S_B^{(0)} \Sigma_B^{(0)} (V_B^{(0)})^\top, \tag{10}$$

We retain the available leading singular components up to the factor dimension N_r, absorb the retained singular values into $S^{(0)}$, and initialize the value side identically. This SVD warm start is motivated by the strong spectral concentration of the learned coefficient matrices (Fig. 1(b)).

E-step. Given the current low-rank factors, we reconstruct the codebook and assign each calibration sub-vector to its nearest codeword, yielding

$$\mathcal{S}_K = \{s_m^i\}, \qquad \mathcal{S}_V = \{z_m^i\}, \tag{11}$$

where s_m^i and z_m^i are the selected indices for the i-th key and value sub-vector of token m. These assignments define the unconstrained targets used in the M-step.

M-step. Given fixed assignments, we first compute the unconstrained optimal coefficient matrices $\bar{A}^\star, \bar{B}^\star \in \mathbb{R}^{D\times N_c}$ that minimize the per-subspace reconstruction error under the current assignments; closed-form entries are given by the per-cluster means of the assigned sub-vectors (see Appendix B.3). Let $w_{i,l} = |\mathcal{C}_l^i|$ denote the number of calibration sub-vectors assigned to codeword l in subspace i. The value side counterparts $\bar{A}_v^\star, \bar{B}_v^\star$ are computed analogously. We then solve the regularized low-rank approximation problems

$$\min_{S_A,V_A} \sum_{i,l} w_{i,l} \left(\bar{A}_{i,l}^\star - (S_A V_A^\top)_{i,l}\right)^2 + \lambda \left(\|S_A\|_F^2 + \|V_A\|_F^2\right), \tag{12}$$

$$\min_{S_B,V_B} \sum_{i,l} w_{i,l} \left(\bar{B}_{i,l}^\star - (S_B V_B^\top)_{i,l}\right)^2 + \lambda \left(\|S_B\|_F^2 + \|V_B\|_F^2\right), \tag{13}$$

where $\lambda > 0$ is a regularization coefficient. Each joint (S, V) problem is bilinear (non-convex), but reduces to a convex regularized least-squares problem with one factor fixed. Symmetrical problems are solved for the value side. Each subproblem is optimized by alternating least squares (ALS) [29]. For example, fixing V_A, each row of S_A has the closed-form update

$$s_{A,i} \leftarrow \left(V_A^\top W_i V_A + \lambda I_{N_r}\right)^{-1} V_A^\top W_i \bar{A}_{i,:}^{\star\top}, \tag{14}$$

where $W_i = \mathrm{diag}(w_{i,:})$ and I_{N_r} is the $N_r \times N_r$ identity matrix. The updates for V_A, S_B, and V_B are obtained analogously, and the same procedure is applied to the value side. Each ALS sub-step does not increase the fixed-assignment regularized objective. After the prescribed EM iterations, we reconstruct the final codebooks using Eqs. (7) and (9), discarding all low-rank factors. The full algorithm and convergence discussion are given in Appendix C and Appendix D. In our default LLaMA-3.1-8B-Instruct setting, the offline codebook construction takes about 6 GPU hours, which is a one-time cost amortized across all subsequent inference runs. Additional measurements on 13B and 32B models are provided in Appendix E.

4.4 Online Quantization and Attention Computation

As shown in Fig. 2, RankVQ consists of an offline codebook construction stage and an online KV cache quantization and attention stage. After offline low-rank parameterization, RankVQ uses the same online pipeline as standard commutative VQ: the cache stores discrete indices, and attention is computed directly from the deployed codebooks during both prefilling and decoding.

Writing the Quantized KV Cache. For a token at position m, we partition its key into D RoPE subspaces, $k_m = [k_m^1, \ldots, k_m^D]$, with $k_m^i \in \mathbb{R}^{1\times 2}$. Each sub-vector is assigned to the nearest subspace-specific vector-form codeword

$$s_m^i = \arg\min_{l\in\{0,\ldots,N_c-1\}} \left\|k_m^i - c_K^{i,l}\right\|_2^2. \tag{15}$$

The key cache stores the resulting discrete assignments $\mathcal{S}_K = \{s_m^i\}_{m,i}$ rather than full-precision key vectors.

This single-stage assignment is used for exposition. The deployed implementation adopts the grouped residual extension described in Appendix B.4. The value side follows the same cache-writing principle: each value sub-vector is quantized to an index z_m^i, and the cache stores $\mathcal{S}_V = \{z_m^i\}_{m,i}$.

Attention Computation with the Quantized Cache. During attention, each cached key index s_m^i is used to retrieve the corresponding codeword C_K^{i,s_m^i} via table lookup. Let $e_1 = [1, 0]$ and define the token-dependent row vector

$$h_m^i = e_1 R_m^i = [\cos(m\theta_i),\ \sin(m\theta_i)] \in \mathbb{R}^{1\times 2}. \tag{16}$$

Because the vector-form codeword is the first row of the pseudo-symmetric matrix, i.e. $c_K^{i,l} = e_1 C_K^{i,l}$, its rotated form satisfies

$$c_K^{i,l} R_m^i = h_m^i C_K^{i,l}, \tag{17}$$

where the equality uses $R_m^i C_K^{i,l} = C_K^{i,l} R_m^i$ from Property 1. For the current query $q_t^i \in \mathbb{R}^{1\times 2}$ in the i-th RoPE subspace, the subspace-wise contribution to the attention score is

$$\alpha_m^i = (q_t^i R_t^i)\big(c_K^{i,s_m^i} R_m^i\big)^\top. \tag{18}$$

Using Eq. (17), we obtain

$$\alpha_m^i = (q_t^i R_t^i)\big(h_m^i C_K^{i,s_m^i}\big)^\top = (q_t^i R_t^i)\big(C_K^{i,s_m^i}\big)^\top (h_m^i)^\top. \tag{19}$$

The per-codeword term

$$\beta_t^{i,l} = (q_t^i R_t^i)\big(C_K^{i,l}\big)^\top \in \mathbb{R}^{1\times 2} \tag{20}$$

is computed once per query step, subspace, and codeword, then reused for all cached tokens with index l; only $(h_m^i)^\top$ remains token-dependent. In the residual setting, the same identity applies to the stage-summed effective codeword in Appendix B.4. The full attention logit is $\alpha_m = \sum_{i=1}^{D} \alpha_m^i$, and the attention output is

$$\mathrm{Attn}(q_t) = \sum_m \mathrm{softmax}_m\left(\frac{\alpha_m}{\sqrt{d}}\right) \cdot \tilde{v}_m, \tag{21}$$

where $\tilde{v}_m$ is obtained by value side codebook lookup. This avoids explicit key reconstruction while leaving value lookup as a simple table access.

4.5 Complexity and Memory Analysis

RankVQ leaves deployed inference unchanged relative to a standard commutative codebook: attention still uses Eq. (19), and the low-rank factors never appear online because they are discarded before deployment. Thus, online complexity depends on deployed codebook sizes and context length, not on the factor dimension N_r.

Additional cost appears only during offline EM/ALS training. One ALS update scales as DN_cN_r plus an $N_r \times N_r$ solve. A detailed breakdown of inference complexity, offline construction complexity, and offline parameterization accounting is provided in Appendix E.

5 Experiments

5.1 Settings

Models. We evaluate RankVQ primarily on LLaMA-3.1-8B-Instruct [5] with context lengths up to 128K. Codebooks are constructed using a subset of FineWeb-Edu [30]; detailed codebook configurations are provided in Appendix B.2. We report results under 2-bit and 1-bit KV cache quantization, corresponding to moderately compressed and ultra-low-bit settings. To examine cross-backbone generalization, we further evaluate RankVQ on LLaMA-2-7B [31] and Mistral-7B [32] in Appendix F.1.

Baselines. We compare RankVQ with representative KV cache compression methods, including KIVI [26], KVQuant [33], VQLLM [34], and CommVQ [19]. These baselines cover scalar or mixed-precision quantization, vector quantization, and RoPE-aware commutative codebooks. Unless otherwise stated, main-text results are reported at comparable average bit-widths around each target compression level. Additional same-budget controls, including settings with identical (N_c, R, g) or identical measured memory footprint, and comparisons with TurboQuant [35] are provided in Appendices F.2 and F.3, respectively. We denote each variant as <method>-<n>, where n indicates the nominal quantization setting; the corresponding deployed average bit-width is reported explicitly in each table.

Tasks. We evaluate long-context performance on LongBench [36] and InfiniteBench [37], and reasoning performance on GSM8K [38] and MATH-500 [39]. LongBench covers diverse long-context tasks such as multi-document question answering, summarization, and code understanding.

Table 1: LongBench evaluation for the LLaMA-3.1-8B-Instruct model.

Methods	Avg. bit (↓)	Qasper	QMSum	MultiNews	TREC	TriviaQA	SAMSum	LCC	RepoBench-P	Average (↑)
FP16 Baseline	16	25.19	23.31	26.82	72.50	91.65	43.49	52.47	49.01	48.05
KIVI-2	3.00	22.71	24.33	27.29	72.50	92.06	43.26	51.32	47.53	47.62
KVQuant-2	2.33	41.86	22.37	25.76	69.00	89.00	42.09	36.22	36.51	45.35
VQLLM-2	2.00	32.39	25.20	26.22	69.95	92.01	41.03	40.58	36.19	45.45
CommVQ-2	2.00	24.67	24.36	26.48	72.50	91.92	43.98	53.02	46.92	47.98
RankVQ-2	**1.64**	**25.30**	**24.80**	**26.90**	**72.50**	**92.10**	**44.20**	**53.30**	**47.30**	**48.30**
KIVI-1	2.00	4.99	9.57	9.20	38.75	25.07	11.93	17.67	16.40	16.70
KVQuant-1	1.33	1.01	8.71	6.06	1.00	1.50	6.64	11.01	11.09	5.88
VQLLM-1	1.00	11.92	17.91	13.12	47.98	63.34	23.72	18.92	22.44	27.42
CommVQ-1	1.03	18.86	23.02	24.34	69.00	91.61	41.83	48.78	42.08	44.94
RankVQ-1	**0.86**	**22.53**	**23.50**	**25.78**	**71.12**	**91.86**	**42.52**	**50.51**	**43.07**	**46.36**

Table 2: InfiniteBench evaluation for the LLaMA-3.1-8B-Instruct model.

Methods	Avg. bit (↓)	R.PK	R.Num	R.KV	En.Sum	En.QA	En.MC	En.Dia	Code.D	Math.F	Average (↑)
FP16 Baseline	16	100.00	99.49	55.20	26.74	14.28	66.81	20.00	22.08	33.43	48.67
KIVI-2	3.00	100.00	97.80	0.60	25.41	13.90	66.81	22.50	23.35	33.71	42.68
KVQuant-2	2.33	98.81	88.81	0.00	25.02	7.77	35.81	8.00	25.63	10.29	33.34
VQLLM-2	2.00	100.00	97.96	0.00	18.27	8.09	44.54	9.50	21.83	29.71	36.66
CommVQ-2	2.00	100.00	93.39	12.20	24.14	14.57	67.25	18.00	22.08	33.14	42.75
RankVQ-2	**1.64**	**100.00**	**99.36**	**15.31**	**26.11**	**14.63**	**67.36**	**22.78**	**23.41**	**33.86**	**44.76**
KIVI-1	2.00	1.86	0.00	0.00	12.44	3.24	55.46	4.50	23.86	34.00	15.04
KVQuant-1	1.33	0.00	0.00	0.00	17.83	1.36	0.44	2.00	0.25	1.14	2.56
VQLLM-1	1.00	82.37	14.75	0.00	10.46	2.69	25.33	1.50	21.83	7.43	18.48
CommVQ-1	1.03	99.15	62.37	0.00	19.34	11.30	65.50	18.00	22.08	33.14	36.76
RankVQ-1	**0.86**	**100.00**	**68.17**	**8.01**	**21.32**	**12.61**	**67.16**	**19.45**	**23.03**	**35.07**	**39.42**

InfiniteBench further stresses ultra-long-context retrieval and reasoning, while GSM8K and MATH-500 evaluate whether KV cache compression preserves mathematical reasoning. In the grouped residual setting used in our experiments, the average bit-width is determined by the group size g and the number of residual stages R. See Appendix B.1 for details.

5.2 Long Context Benchmarks Evaluation

LongBench Evaluation. Table 1 reports the LongBench results under comparable average bit-widths. RankVQ achieves the highest average score among compressed variants while using a lower average bit-width than prior commutative-codebook baselines. RankVQ-2 reaches 48.30 at 1.64 Avg. bit, numerically matching the FP16 baseline, and RankVQ-1 reaches 46.36 at 0.86 Avg. bit, yielding the leading result among 1-bit variants. Because the compared methods are evaluated at nearby rather than exactly identical operating points, we do not claim strict same-budget dominance from this table alone. Instead, the results indicate a more favorable accuracy–compression trade-off, with the same-budget controls provided in Appendix F.2.

InfiniteBench Evaluation. Table 2 shows a consistent trend on more challenging ultra-long-context tasks. RankVQ outperforms the compared baselines at lower average bit-widths, suggesting that its structured codebook construction improves the quality of commutative codebooks under aggressive compression. On the adversarial R.KV retrieval task, all compared 1-bit KV quantization baselines collapse to 0.00, whereas RankVQ-1 retains a score of 8.01. This result indicates that RankVQ better preserves retrieval performance in the ultra-low-bit regime.

Needle-in-a-Haystack Evaluation. Fig. 3 compares RankVQ with CommVQ [19] on the Needle-in-a-Haystack benchmark, which evaluates retrieval fidelity across different context lengths and insertion depths. RankVQ-1 remains more stable than CommVQ-1 as the context length increases and the insertion position becomes more challenging, with clear advantages in the 64K–128K regime. The corresponding 2-bit results in Appendix F.5 provide further evidence that RankVQ improves robustness in extremely long-context retrieval.

5.3 Reasoning Evaluation

GSM8K Evaluation. Table 3 evaluates whether KV cache compression preserves multi-step mathematical reasoning on GSM8K. RankVQ-2 achieves 76.54 at 1.64 Avg. bit, numerically matching the FP16 baseline. Under the more aggressive 1-bit setting, RankVQ-1 reaches 70.07 at 0.86 Avg. bit and outperforms the reported 1-bit baselines. These results suggest that the codebook constructed

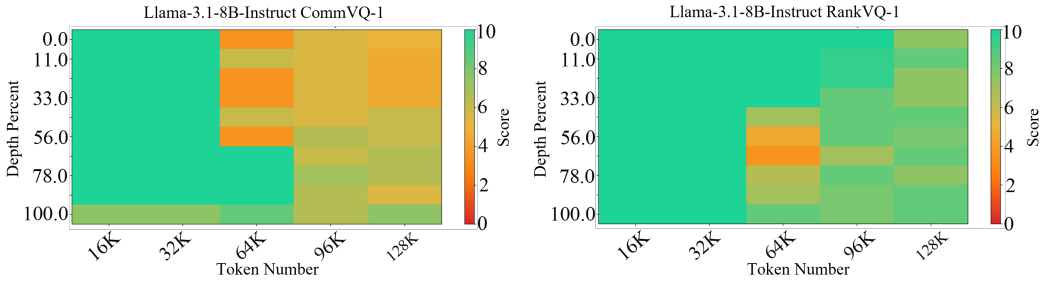

Figure 3: Needle-in-a-Haystack results across context lengths and insertion depths. Higher scores indicate better retrieval fidelity.

Table 3: GSM8K benchmark evaluation for the LLaMA-3.1-8B-Instruct model.

Methods	FP16 Baseline	KIVI-2	VQLLM-2	CommVQ-2	**RankVQ-2**	KIVI-1	VQLLM-1	CommVQ-1	**RankVQ-1**
Avg. bit (↓)	16	3.00	2.00	2.00	**1.64**	2.00	1.00	1.03	**0.86**
GSM8K (↑)	76.27	73.69	52.69	76.04	**76.54**	2.20	1.67	66.57	**70.07**

by RankVQ preserves information useful for reasoning, not only for long-context retrieval and understanding.

MATH-500 Evaluation. We further evaluate the 1-bit setting on MATH-500, which contains harder problems with longer generated solutions. We rank the 500 problems by the number of tokens generated under FP16 decoding and divide them into three approximately equal-sized groups. RankVQ-1 improves over CommVQ-1 by 2.20, 3.21, and 4.79 points on the short, medium, and long groups, respectively. The widening gap indicates that RankVQ better preserves reasoning performance for problems with longer generated solutions. Full results are provided in Appendix F.6.

5.4 Ablation Studies

Pipeline ablation. Table 4 ablates the main components of RankVQ under the same RankVQ-1 deployment configuration. We compare five variants: (i) **B**, a full-rank commutative VQ baseline trained with the same deployment configuration; (ii) **B+SVD**, which applies truncated SVD to the full-rank reference codebook parameters and deploys the reconstructed codebook without EM refinement; (iii) **B+SVD+EM**†, which performs low-rank EM refinement only for the key-side parameters while keeping the value codebook full-rank; (iv) **B+SVD+EM***, which performs low-rank EM refinement only for the value-side parameters while keeping the key codebook full-rank; and (v) **B+SVD+EM**, the full RankVQ pipeline, which applies low-rank EM refinement to the parameters underlying both key and value codebooks. At the default $N_r = N_c = 32$, rank-32 SVD reconstructs the full-rank reference up to numerical precision, so the difference between B and B+SVD should not be interpreted as an improvement produced by SVD (Appendix F.8). The further improvement from B+SVD to B+SVD+EM† shows that assignment-aware EM refinement is a key source of performance gain. The value-side-only B+SVD+EM* variant achieves a similar improvement, showing that value-side refinement also provides a substantial benefit. The full B+SVD+EM variant achieves the best result, demonstrating the benefit of applying low-rank EM refinement to both key and value codebooks. Matched-budget comparisons are provided in Appendix F.2.

Table 4: Pipeline ablation study on LongBench under the RankVQ-1 configuration.

Variants	Avg. bit	LongBench (↑)
B	0.86	42.37
B+SVD	0.86	43.25
B+SVD+EM†	0.86	45.71
B+SVD+EM*	0.86	45.68
B+SVD+EM (Ours)	0.86	**46.36**

Table 5: Sensitivity to the factor dimension N_r under 1-bit quantization on LLaMA-3.1-8B-Instruct.

Variants	N_r	LongBench (↑)
RankVQ-1	8	43.82
RankVQ-1	16	45.61
RankVQ-1 (default)	32	**46.36**
RankVQ-1	64	46.22

Factor-dimension sensitivity. Table 5 studies the effect of the factor dimension N_r under the default 1-bit setting. Very small factor dimensions noticeably degrade performance, suggesting that an overly restrictive factorization cannot sufficiently capture codebook variation. In contrast, moderate factor dimensions recover most of the benefit, and performance becomes largely stable once N_r reaches the default value. This indicates that RankVQ is not highly sensitive to the precise factor dimension within a reasonable range.

Additional ablations on the regularization coefficient λ, factor initialization, calibration-sample variance, and robustness are presented in Appendix F.8.

5.5 Efficiency Results

Table 6 reports decoding efficiency and deployed KV cache footprint at a 128K context length with batch size 1. For reproducibility, each compressed baseline is evaluated under its default 1-bit configuration. RankVQ-1 achieves the smallest deployed cache, reducing KV cache memory from 16.80 GB in FP16 to 1.44 GB. In comparison, KIVI-1 [26] and CommVQ-1 [19] require 2.98 GB and 2.45 GB, respectively. RankVQ-1 also achieves the lowest decoding latency among compressed baselines. It obtains 65.37 ms/token, compared with 75.42 ms/token for KIVI-1 and 70.60 ms/token for CommVQ-1. RankVQ-1 also operates at a lower effective average bit-width than CommVQ-1 (0.86 vs. 1.03) while preserving the same commutative attention kernel form. These results indicate that RankVQ improves the deployed memory–latency trade-off of KV cache compression, rather than only improving offline codebook quality. More detailed efficiency results are provided in Appendix F.9.

Table 6: Efficiency comparison at batch size 1 for LLaMA-3.1-8B with 128K context length.

Context	Methods	Latency (ms/token) ↓	Throughput (token/s) ↑	KV cache (GB) ↓
128K	FP16 Baseline	63.78	15.68	16.80
	KIVI-1	75.42	13.26	2.98
	CommVQ-1	70.60	14.16	2.45
	RankVQ-1	**65.37**	**15.30**	**1.44**

6 Conclusion

We presented RankVQ, a low-rank parameterized commutative vector quantization method for KV cache compression in long-context LLMs. RankVQ exploits parameter-side redundancy in the scalar coefficient matrices used to construct RoPE-aware commutative codebooks. By factorizing these coefficients and refining them through assignment-aware EM, RankVQ improves codebook construction while preserving the pseudo-symmetric structure required for reconstruction-free commutative attention. The low-rank factors are used only during offline codebook construction and are discarded before deployment, leaving the online inference path unchanged. Experiments on LongBench, InfiniteBench, and GSM8K show that RankVQ improves the accuracy–compression trade-off over directly comparable KV quantization baselines, with especially clear gains under ultra-low-bit compression. These results highlight the value of improving codebook construction for efficient long-context LLM inference.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No. 2025YFC3409900), the Tianjin Natural Science Foundation Project (No. 25JCYBJC01540), the Tianjin Municipal Science and Technology Major Program (No. 25ZXZSSS00390), the Tianjin Future Industry Science and Technology Major Project (No. 25ZXWCSY00290), the Emerging Frontiers Cultivation Program of Tianjin University Interdisciplinary Center, and the Independent Research Project of State Key Laboratory of Synthetic Biology (No. HCZC-202605A). The authors declare no competing interests.

References

[1] Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. Wordcraft: Story writing with large language models. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*, pages 841–852. Association for Computing Machinery, 2022.

[2] Wenjie Dong, Sirong Chen, and Yan Yang. ProTOD: Proactive task-oriented dialogue system based on large language model. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9147–9164. Association for Computational Linguistics, 2025.

[3] Qiuhan Gu. LLM-based code generation method for Golang compiler testing. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 2201–2203. Association for Computing Machinery, 2023.

[4] Menglin Yang, Aosong Feng, Bo Xiong, Jiahong Liu, Irwin King, and Rex Ying. Enhancing LLM complex reasoning capability through hyperbolic geometry. In *ICML 2024 Workshop on LLMs and Cognition*, 2024.

[5] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[6] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. LongRoPE: Extending LLM context window beyond 2 million tokens. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11091–11104. PMLR, 2024.

[7] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. LLM maybe LongLM: SelfExtend LLM context window without tuning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 22099–22114. PMLR, 2024.

[8] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31094–31116. PMLR, 2023.

[9] Yue Zheng, Yuhao Chen, Bin Qian, Xiufang Shi, Yuanchao Shu, and Jiming Chen. A review on edge large language models: Design, execution, and applications. *ACM Computing Surveys*, 57 (8):1–35, 2025.

[10] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 34661–34710, 2023.

[11] Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. ArkVale: Efficient generative LLM inference with recallable key-value eviction. In *Advances in Neural Information Processing Systems*, volume 37, pages 113134–113155, 2024.

[12] Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. NACL: A general and effective KV cache eviction framework for LLM at inference time. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7913–7926. Association for Computational Linguistics, 2024.

[13] Shang Yang, Junxian Guo, Haotian Tang, Qinghao Hu, Guangxuan Xiao, Jiaming Tang, Yujun Lin, Zhijian Liu, Yao Lu, and Song Han. LServe: Efficient long-sequence LLM serving with unified sparse attention. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

[14] Qianchao Zhu, Jiangfei Duan, Chang Chen, Siran Liu, Xiuhong Li, Guanyu Feng, Xin Lv, Xiao Chuanfu, Dahua Lin, and Chao Yang. SampleAttention: Near-lossless acceleration of long context LLM inference with adaptive structured sparse attention. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

[15] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems*, volume 37, pages 52481–52515, 2024.

[16] Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. Keep the cost down: A review on methods to optimize LLM's KV-cache consumption. In *First Conference on Language Modeling*, 2024.

[17] Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu, Zirui Liu, and Xia Hu. KV cache compression, but what must we give in return? a comprehensive benchmark of long context capable approaches. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 4623–4648. Association for Computational Linguistics, 2024.

[18] Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, Zhen Dong, Animashree Anandkumar, Abedelkadir Asi, and Junjie Hu. R-KV: Redundancy-aware KV cache compression for reasoning models. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

[19] Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Colorado Reed, Chong Wang, and Chuang Gan. CommVQ: Commutative vector quantization for KV cache compression. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 36831–36845. PMLR, 2025.

[20] Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12284–12303. PMLR, 2024.

[21] Donghyun Son, Euntae Choi, and Sungjoo Yoo. NSNQuant: A double normalization approach for calibration-free low-bit vector quantization of KV cache. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

[22] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[23] William Timkey and Marten van Schijndel. All bark and no bite: Rogue dimensions in transformer language models obscure representational quality. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4527–4546. Association for Computational Linguistics, 2021.

[24] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[25] Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen Attention: Attention in low-rank space for KV cache compression. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 15332–15344. Association for Computational Linguistics, 2024.

[26] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 32332–32344. PMLR, 2024.

[27] Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pages 57355–57373. PMLR, 2025.

[28] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[29] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the Netflix Prize. In *Algorithmic Aspects in Information and Management*, volume 5034 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2008.

[30] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, volume 37, pages 30811–30849, 2024.

[31] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[32] 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. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.

[33] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, volume 37, pages 1270–1303, 2024.

[34] Ankur Kumar. Residual vector quantization for KV cache compression in large language model. In *Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop*, volume 262 of *Proceedings of Machine Learning Research*, pages 485–490. PMLR, 2024.

[35] Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, volume 2026, pages 56418–56439, 2026.

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

[37] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞Bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277. Association for Computational Linguistics, 2024.

[38] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[39] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601, 2024.

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

[41] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 38087–38099. PMLR, 2023.

[42] Yuxin Chen, Yuejie Chi, Jianqing Fan, and Cong Ma. Gradient descent with random initialization: Fast global convergence for nonconvex phase retrieval. *Mathematical Programming*, 176 (1–2):5–37, 2019.

[43] Serge L Shishkin, Arkadi Shalaginov, and Shaunak D Bopardikar. Fast approximate truncated SVD. *Numerical Linear Algebra with Applications*, 26(4):e2246, 2019.

A Related Work

KV cache compression for long-context inference. Efficient long-context inference has motivated a broad line of work on reducing KV cache memory and computation costs. Existing methods mainly follow three directions: token eviction, sparse attention, and KV cache quantization. Token eviction reduces memory usage by removing less important cached tokens. For example, H_2O [10] identifies heavy-hitter tokens and discards low-attention entries, while ArkVale [11] introduces recallable eviction so that evicted tokens can be retrieved when needed. Sparse attention instead lowers computation cost by restricting attention to a subset of key-value pairs. LServe [13] proposes a unified sparse attention framework with hardware-friendly structured sparsity for both prefilling and decoding, and SampleAttention [14] dynamically selects important key-value pairs through query-guided filtering. These methods are effective for reducing inference cost, but they may lose information needed by tasks that require faithful full-context access. In contrast, RankVQ keeps all cached tokens and compresses the representation of each token itself.

Vector quantization for LLM and KV cache compression. KV cache quantization provides a complementary way to reduce memory footprint by compressing cached key-value representations while preserving the complete token sequence. Among quantization methods, vector quantization (VQ) is especially attractive in ultra-low-bit settings because it represents continuous vectors with compact discrete indices. AQLM [20] applies additive vector quantization for extreme LLM weight compression, while NSNQuant [21] introduces a calibration-free double-normalization strategy for low-bit KV cache vector quantization. VQLLM [34] further adopts residual vector quantization for KV cache compression. More closely related to our work, CommVQ [19] exploits the algebraic structure of RoPE and constructs pseudo-symmetric commutative codebooks, allowing attention scores to be computed without explicit key reconstruction. RankVQ keeps the same commutative inference form as CommVQ, but improves the codebook parameterization by imposing low-rank structure on the underlying scalar coefficients.

Low-rank factorization. Low-rank factorization has been widely used in efficient LLM adaptation and compression, including parameter-efficient fine-tuning [28], weight compression [25], and activation or KV cache compression [26, 27]. RankVQ differs from these methods in both its target and deployment behavior. Rather than directly factorizing model weights, activations, KV tensors, or the deployed codebook, RankVQ imposes a low-rank structure on the scalar coefficient matrices used to construct commutative codebooks. This low-rank parameterization is optimized only during offline codebook construction and is discarded before deployment. Therefore, RankVQ improves codebook construction quality and the compression–accuracy trade-off while leaving the deployed inference procedure unchanged.

B Implementation details

B.1 Deployed Bit-Width Accounting

We use Avg. bit to measure the analytical number of stored index bits per cached scalar. Following [19], we group every g consecutive 2D sub-vectors into one group and let all sub-vectors in the group share the same quantized assignment. Since each 2D sub-vector contains two FP16 scalars, each group corresponds to $2g$ scalars. Under the residual vector quantization setting used in this paper, the quantization procedure is repeated for R stages, yielding R codebooks. In each stage, one discrete codeword index is stored for each group, and each index requires $\log_2(N_c)$ bits when the stage codebook contains N_c codewords. Therefore, quantizing one group over R stages requires $R\log_2(N_c)$ bits. Since a d-dimensional vector contains $d/(2g)$ such groups, the total number of bits required for the full vector is

$$R\log_2(N_c)\cdot\frac{d}{2g}. \tag{22}$$

Accordingly, the average deployed bit width per scalar is

$$\text{Avg. bit} = \frac{R\log_2(N_c)}{2g}. \tag{23}$$

Here, N_r is used only during offline codebook construction and controls the intermediate representation capacity of the low-rank parameterization. It is not stored in the deployed KV cache and therefore

does not appear in the deployed bit-width accounting. The low-rank factors introduced during offline optimization are discarded before deployment, and only the final codebooks and discrete assignments are retained at inference time.

B.2 Codebook Configuration

Unless otherwise specified, the default configuration in all main-text experiments is as follows. We use the same codebook configuration for both key and value caches, with $N_v = N_c$. For the 2-bit setting, we set the codebook size to $N_c = 32$, the number of quantization stages to $R = 21$, and the group size to $g = 32$. For the 1-bit setting, we set $N_c = 32$, $R = 11$, and $g = 32$. The factor dimension is $N_r = 32$ for both keys and values. The regularization coefficient is $\lambda = 10^{-3}$. The EM algorithm runs for $T_{\mathrm{EM}} = 10$ outer iterations, with $T_{\mathrm{ALS}} = 5$ inner ALS iterations per M-step. Calibration uses a FineWeb-Edu [30] subset of 1M tokens. Quantization is applied uniformly across all attention heads and layers. Configurations used in the ablation experiments of Appendix F.8 override a subset of these defaults as indicated there.

B.3 M-step unconstrained optimum

Given fixed assignments $\mathcal{S}_K = \{s_m^i\}$, the unconstrained coefficient matrices $\bar{A}^\star, \bar{B}^\star \in \mathbb{R}^{D \times N_c}$ that minimize the per-subspace squared reconstruction error

$$\sum_{m,i} \left\| k_m^i - [\bar{A}_{i,s_m^i},\ \bar{B}_{i,s_m^i}] \right\|_2^2 \tag{24}$$

are given, for non-empty clusters, by the per-cluster means of the assigned sub-vectors:

$$\bar{A}_{i,l}^\star = \frac{1}{|\mathcal{C}_l^i|} \sum_{m \in \mathcal{C}_l^i} (k_m^i)_1, \qquad \bar{B}_{i,l}^\star = \frac{1}{|\mathcal{C}_l^i|} \sum_{m \in \mathcal{C}_l^i} (k_m^i)_2, \tag{25}$$

where $\mathcal{C}_l^i = \{m : s_m^i = l\}$ is the cluster of calibration sub-vectors assigned to codeword l in RoPE subspace i, and $(k_m^i)_k$ denotes the k-th scalar entry of k_m^i. The value side counterparts $\bar{A}_v^\star, \bar{B}_v^\star$ are computed analogously. For an empty cluster, $w_{i,l} = 0$ and the corresponding data-fitting term vanishes; in implementation, we retain the previous iteration's codeword entry.

B.4 Residual and Grouped Extension Used in Experiments

Sections 4.4 and 5 present the single-stage, per-subvector form for readability. The actual deployment used in all reported experiments follows residual vector quantization with one shared assignment per group of g consecutive 2D sub-vectors, following the CommVQ deployment protocol. Let $\mathcal{G}(u) \subset \{1, \ldots, D\}$ denote group u with $|\mathcal{G}(u)| = g$, and let $\rho_{m,1}^i = k_m^i$ be the initial key residual for token m and RoPE subspace i. For stage $r \in \{1, \ldots, R\}$, the stage-specific grouped assignment is

$$s_{m,r}^u = \arg \min_{l \in \{0,\ldots,N_c-1\}} \sum_{i \in \mathcal{G}(u)} \left\| \rho_{m,r}^i - c_{K,r}^{i,l} \right\|_2^2, \tag{26}$$

and the residual is updated elementwise inside the group as

$$\rho_{m,r+1}^i = \rho_{m,r}^i - c_{K,r}^{i,\, s_{m,r}^u}, \qquad i \in \mathcal{G}(u). \tag{27}$$

Value side assignments $z_{m,r}^u$ are defined analogously. The effective deployed codeword in subspace i is the stage sum $\hat{c}_{K,m}^i = \sum_{r=1}^R c_{K,r}^{i,\, s_{m,r}^{u(i)}}$, with the associated stage-summed pseudo-symmetric matrix $\hat{C}_{K,m}^i = \sum_{r=1}^R C_{K,r}^{i,\, s_{m,r}^{u(i)}}$. Since each stage-specific matrix is pseudo-symmetric, their sum remains pseudo-symmetric and commutes with the corresponding RoPE rotation. The single-stage equations in Section 4 are recovered by setting $R = 1$ and $g = 1$. This grouped residual extension changes deployed bit accounting and assignment tying, but it does not change the low-rank parameterization itself, which is learned stage-wise by the same EM/ALS procedure.

C Low-Rank Parameterized EM Algorithm

Algorithm 1 summarizes the low-rank parameterized EM procedure used for offline codebook construction. We first obtain a full-rank commutative codebook as a reference solution, initialize the low-rank factors by truncated SVD, and then alternate between nearest-codeword assignment and regularized alternating least squares [29] updates to refine the low-rank parameter matrices.

Algorithm 1 RankVQ offline codebook construction.

Require: Calibration pre-RoPE keys $\{k_m\}$ and values $\{v_m\}$; factor dimension N_r; codebook sizes N_c, N_v; regularization λ; iterations $T_{\mathrm{EM}}, T_{\mathrm{ALS}}$.

Construct a full-rank commutative codebook to obtain reference matrices $A_{\mathrm{orig}}, B_{\mathrm{orig}}$ (key) and $A_{v,\mathrm{orig}}, B_{v,\mathrm{orig}}$ (value).
Initialize the low-rank factors by truncated SVD using factor dimension N_r, absorbing the retained singular values into the corresponding S matrices.
for $t = 1, \ldots, T_{\mathrm{EM}}$ **do**
    E-step: Reconstruct codebooks via Eqs. (7)–(9); assign each calibration sub-vector to its nearest codeword to obtain $\mathcal{S}_K = \{s_m^i\}$ and $\mathcal{S}_V = \{z_m^i\}$.
    Given the current assignments, compute the corresponding unconstrained optimal coefficient matrices $\bar{A}^\star, \bar{B}^\star, \bar{A}_v^\star$, and $\bar{B}_v^\star$, together with the assignment occupancies $w_{i,l}$.
    for $\tau = 1, \ldots, T_{\mathrm{ALS}}$ **do** ▷ **M-step**
        Update (S_A, V_A) and (S_B, V_B) by weighted ALS; Eq. (14) gives the representative closed-form update.
        Update (S_{A_v}, V_{A_v}) and (S_{B_v}, V_{B_v}) analogously for the value side.
    end for
end for
Reconstruct the deployed codebooks via Eqs. (7)–(9).
Discard all low-rank factors and **return** the deployed key and value codebooks.

D Convergence Analysis

Monotone Descent of the M-step. We present the argument for the key-side coefficient matrix A; the same argument applies to B and to the value-side subproblems. For fixed assignments, let $w_{i,l} = |\mathcal{C}_l^i|$. The sample-level reconstruction term admits the exact decomposition

$$\sum_{m,i} \left((k_m^i)_1 - A_{i,s_m^i}\right)^2 = \sum_{i,l} w_{i,l} \left(A_{i,l} - \bar{A}_{i,l}^\star\right)^2 + C(\mathcal{S}), \tag{28}$$

where $C(\mathcal{S})$ is independent of A. Therefore, the fixed-assignment low-rank subproblem is

$$\mathcal{L}_A(S_A, V_A) = \sum_{i,l} w_{i,l} \left(\bar{A}_{i,l}^\star - (S_A V_A^\top)_{i,l}\right)^2 + \lambda \left(\|S_A\|_F^2 + \|V_A\|_F^2\right). \tag{29}$$

Fix V_A and consider minimization over S_A. The problem separates over the rows of S_A. For row i, the objective is

$$\mathcal{L}_{A,i}(s_{A,i}) = \left\| W_i^{1/2} \left(\bar{A}_{i,:}^{\star\top} - V_A s_{A,i}\right) \right\|_2^2 + \lambda \|s_{A,i}\|_2^2 + \mathrm{const}, \tag{30}$$

where $W_i = \mathrm{diag}(w_{i,:})$. This is a strictly convex quadratic because $\lambda > 0$. Setting its gradient to zero gives

$$s_{A,i} = \left(V_A^\top W_i V_A + \lambda I_{N_r}\right)^{-1} V_A^\top W_i \bar{A}_{i,:}^{\star\top}, \tag{31}$$

which is the update in Eq. (14). The update for V_A follows analogously by fixing S_A. Each ALS substep is the exact minimizer in its active variable and therefore does not increase the fixed-assignment regularized objective. The same argument applies to B and to the value-side subproblems.

Non-increasing Outer EM Objective. Let $\mathcal{S}$ denote the key and value assignments, and define the fixed-assignment regularized objective

$$\mathcal{F}_\lambda(\Theta, \mathcal{S}) = \sum_{m,i} \left\| k_m^i - c_K^{i,s_m^i}(\Theta) \right\|_2^2 + (\text{value-side term}) + \lambda R(\Theta), \tag{32}$$

where $\Theta = \{S_A, V_A, S_B, V_B, S_{A_v}, V_{A_v}, S_{B_v}, V_{B_v}\}$, and $R(\Theta)$ is the sum of the squared Frobenius norms of the factors. Define the assignment-minimized objective as

$$\mathcal{J}_\lambda(\Theta) = \min_{\mathcal{S}} \mathcal{F}_\lambda(\Theta, \mathcal{S}). \tag{33}$$

At outer iteration t, the E-step selects

$$\mathcal{S}^{(t)} \in \arg\min_{\mathcal{S}} \mathcal{F}_\lambda(\Theta^{(t)}, \mathcal{S}). \tag{34}$$

Hence, for every Θ,

$$\mathcal{J}_\lambda(\Theta) \le \mathcal{F}_\lambda(\Theta, \mathcal{S}^{(t)}), \tag{35}$$

and the bound is tight at the current iterate:

$$\mathcal{F}_\lambda(\Theta^{(t)}, \mathcal{S}^{(t)}) = \mathcal{J}_\lambda(\Theta^{(t)}). \tag{36}$$

Since the weighted M-step starts from the current factors and does not increase the fixed-assignment objective,

$$\mathcal{F}_\lambda(\Theta^{(t+1)}, \mathcal{S}^{(t)}) \le \mathcal{F}_\lambda(\Theta^{(t)}, \mathcal{S}^{(t)}). \tag{37}$$

Combining these relations gives

$$\mathcal{J}_\lambda(\Theta^{(t+1)}) \le \mathcal{F}_\lambda(\Theta^{(t+1)}, \mathcal{S}^{(t)}) \le \mathcal{F}_\lambda(\Theta^{(t)}, \mathcal{S}^{(t)}) = \mathcal{J}_\lambda(\Theta^{(t)}). \tag{38}$$

Therefore, $\mathcal{J}_\lambda(\Theta^{(t)})$ is monotonically non-increasing. Since it is bounded below, its objective-value sequence converges. This argument establishes monotone objective descent and objective-value convergence, but does not imply global optimality.

E Detailed Complexity and Memory Analysis

Inference complexity. After offline codebook construction, RankVQ deploys the same pseudo-symmetric 2×2 key codewords as standard commutative codebooks, together with the corresponding value codebook. Attention is computed using the commutative form in Eq. (19). Therefore, the online decoding cost depends on the deployed codebook sizes N_c and N_v, the number of RoPE subspaces D, and the context length, but not on the factor dimension N_r. In particular, no matrix products involving S_A, S_B, V_A, or V_B appear during decoding; the same holds for their value side counterparts.

Offline construction complexity. The low-rank factors are optimized only during offline codebook construction. In the M-step, RankVQ performs regularized ALS. Let T_{EM} denote the number of EM iterations and T_{ALS} denote the number of ALS iterations within each M-step. The dominant algebraic cost of updating the low-rank factors scales with the size of the coefficient matrix, DN_c, and the factor dimension N_r, together with the solution of $N_r \times N_r$ linear systems. This overhead is paid once during offline construction, while the deployment-time computation remains unchanged.

Measured construction cost across model scales. To assess the offline construction cost beyond the 8B setting, we measure RankVQ on LLaMA-2-13B and Qwen2.5-32B under the same 1M-token calibration set and the same RankVQ configuration. All measurements use a single NVIDIA H100-80GB GPU. Table 7 reports the construction time and peak GPU memory.

Table 7: Offline codebook construction cost across model scales.

Model	Layers	KV heads	D	GPU-h	Peak memory (GB)
LLaMA-3.1-8B-Instruct	32	8	64	6.0	≈ 20
LLaMA-2-13B	40	40	64	32.7	≈ 31
Qwen2.5-32B	64	8	64	14.4	≈ 72

The construction time does not scale directly with model parameter count. For fixed RankVQ hyperparameters, the cost depends mainly on the number of layer-wise KV codebooks, which is determined by the number of layers, KV heads, and RoPE subspaces. This explains why LLaMA-2-13B requires more construction time than Qwen2.5-32B despite having fewer model parameters.

The construction is performed once for each model and codebook configuration, and the low-rank factors are discarded before deployment. A complete 70B construction run is beyond the single-GPU memory setting used here, so we do not extrapolate an unverified runtime.

Offline parameterization accounting. A full commutative key codebook is specified by two scalar coefficient matrices of size $D \times N_c$, requiring $2DN_c$ scalar parameters during codebook construction. RankVQ instead represents these coefficients using four low-rank factors, with total parameter count

$$2DN_r + 2N_cN_r = 2N_r(D + N_c). \tag{39}$$

Thus, the low-rank parameterization uses fewer scalar degrees of freedom whenever $N_r < \frac{DN_c}{D+N_c}$. An analogous accounting applies to the value side by replacing (D, N_c) with (D_v, N_v). At the default $N_r = 32 = \min(D, N_c)$, this parameter-count reduction does not apply. The hard rank constraint is inactive, while the factor regularization remains active as discussed in Section 4.1. The factors are used only during offline codebook construction and are discarded afterward. Therefore, their parameter count does not affect the deployed KV cache or the online inference path.

F Extended results

F.1 Cross-backbone generalization

Table 8 evaluates RankVQ on LLaMA-2-7B [31] and Mistral-7B [32], in addition to LLaMA-3.1-8B-Instruct [5]. The performance gains are consistent across the evaluated backbones, indicating that the benefit is not specific to LLaMA-3.1-8B-Instruct.

Table 8: Cross-backbone evaluation on LLaMA-2-7B (left) and Mistral-7B (right)

Method	**Avg. bit (↓)**	**LongBench (↑)**	**InfiniteBench (↑)**
FP16 Baseline	16	26.30	19.35
VQLLM-2	2.00	22.80	14.10
CommVQ-2	2.00	24.17	17.39
RankVQ-2	**1.64**	**26.91**	**20.06**
VQLLM-1	1.00	10.31	4.17
CommVQ-1	1.03	16.72	9.51
RankVQ-1	**0.86**	**20.39**	**13.23**

Method	**Avg. bit (↓)**	**LongBench (↑)**	**InfiniteBench (↑)**
FP16 Baseline	16	51.87	33.26
VQLLM-2	2.00	46.55	25.18
CommVQ-2	2.00	49.82	29.07
RankVQ-2	**1.64**	**51.96**	**33.47**
VQLLM-1	1.00	28.41	9.78
CommVQ-1	1.03	39.66	20.15
RankVQ-1	**0.86**	**45.37**	**25.06**

F.2 Controlled Budget Comparisons

To separate algorithmic gains from operating-point differences, Table 9 reports controlled comparisons where CommVQ and RankVQ use identical (N_c, R, g) configurations, or identical measured deployed memory in the final row.

Table 9: Controlled budget comparisons between CommVQ and RankVQ.

Protocol	Metric	Avg. bit	CommVQ	RankVQ
Same (N_c, R, g), 2-bit config	LongBench	1.64	47.45	**48.30**
Same (N_c, R, g), 2-bit config	InfiniteBench	1.64	42.10	**44.76**
Same (N_c, R, g), 1-bit config	LongBench	0.86	42.37	**46.36**
Same (N_c, R, g), 1-bit config	InfiniteBench	0.86	30.11	**39.42**
Same deployed memory	GSM8K	0.86	55.44	**70.07**

F.3 Comparison with TurboQuant

Table 10 compares RankVQ with TurboQuant [35] under the same LLaMA-3.1-8B-Instruct backbone. Accuracy follows the same evaluation protocols as the main experiments. Efficiency is measured at 128K context with batch size 1 on a single NVIDIA H100-80GB GPU under the same profiling protocol.

The comparison shows a task-dependent accuracy–compression trade-off. RankVQ-2 achieves 48.30 on LongBench and 76.54 on GSM8K at 1.64 average bits, while TurboQuant-3.5 achieves

Table 10: Accuracy and 128K deployment-efficiency comparison with TurboQuant.

Method	Avg. bit	LongBench	InfiniteBench	GSM8K	KV cache (GB)	Latency (ms/token)	Throughput (token/s)
FP16	16.00	48.05	48.67	76.27	16.80	63.78	15.68
TurboQuant-3.5	3.50	48.05	48.21	75.93	4.06	76.51	13.07
TurboQuant-2.5	2.50	47.46	47.69	74.81	2.94	79.70	12.55
RankVQ-2	1.64	48.30	44.76	76.54	2.38	68.07	14.69
RankVQ-1	0.86	46.36	39.42	70.07	1.44	65.37	15.30

higher InfiniteBench accuracy at 3.50 bits. RankVQ also uses a smaller deployed KV cache and has lower decoding latency than the evaluated TurboQuant configurations. These results extend the accuracy–memory trade-off of RankVQ into the sub-2-bit regime.

F.4 Compatibility with Weight and Activation Quantization

To assess compatibility with model quantization, we combine RankVQ-2 with AWQ [40] W4A16 weight quantization and SmoothQuant [41] W8A8 weight–activation quantization on LLaMA-3.1-8B-Instruct. Table 11 reports the resulting performance on LongBench, InfiniteBench, and GSM8K. The RankVQ codebooks constructed from the FP16 backbone are directly applied to the KV outputs of the quantized model, without additional codebook reconstruction.

The joint results closely follow the corresponding single-component results. Combining RankVQ-2 with AWQ or SmoothQuant introduces little additional accuracy degradation beyond the effects of the individual quantizers. The methods operate on different parts of inference: AWQ and SmoothQuant quantize model weights or activations, while RankVQ quantizes the resulting keys and values stored in the KV cache.

Table 11: Compatibility of RankVQ-2 with weight and activation quantization on LLaMA-3.1-8B-Instruct.

Configuration	Weight	Activation	KV cache	LongBench	InfiniteBench	GSM8K
FP16	W16	A16	FP16	48.05	48.67	76.27
RankVQ-2	W16	A16	RankVQ-2	48.30	44.76	76.54
AWQ	W4	A16	FP16	47.67	48.28	75.66
SmoothQuant	W8	A8	FP16	47.81	48.43	75.89
AWQ + RankVQ-2	W4	A16	RankVQ-2	47.91	44.40	75.93
SmoothQuant + RankVQ-2	W8	A8	RankVQ-2	48.06	44.54	76.16

F.5 Needle-in-a-Haystack at 2-bit

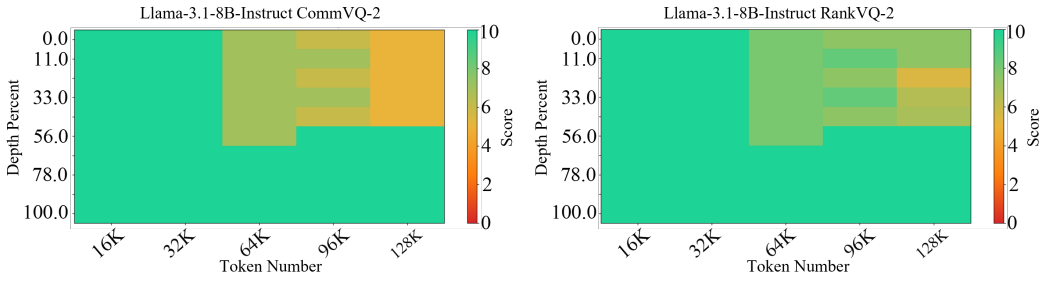

Figure 4: Needle-in-a-Haystack visualization for CommVQ-2 and RankVQ-2 under 2-bit quantization.

Fig. 4 presents the Needle-in-a-Haystack visualization for the 2-bit setting on LLaMA-3.1-8B-Instruct. While both CommVQ-2 and RankVQ-2 achieve nearly lossless retrieval at shorter context lengths, RankVQ-2 appears more robust in the hardest long-context cases. At 96K and 128K tokens, RankVQ-2 tends to preserve higher scores across challenging insertion depths, whereas CommVQ-2 exhibits visibly larger degradation. We interpret this figure as qualitative evidence that is consistent with the benchmark averages in Tables 1 and 2, rather than as a standalone definitive claim.

F.6 MATH-500 Evaluation

We further evaluate reasoning performance on MATH-500 under the 1-bit setting. We rank the 500 problems by the number of tokens generated under FP16 decoding and divide them into three approximately equal-sized groups. The same groups are used for all methods. Table 12 reports the accuracy within each generation-length group and the overall result.

Table 12: MATH-500 accuracy by FP16 generation length under the 1-bit setting.

Length	FP16	CommVQ-1	RankVQ-1	Gain
Short	52.10	47.50	49.70	+2.20
Medium	45.78	36.95	40.16	+3.21
Long	39.52	27.15	31.94	+4.79
Overall	45.80	37.20	40.60	+3.40

F.7 Stage-Wise Low-Rank Structure

We examine whether the low-rank structure is confined to the first residual stage. Under the default 1-bit configuration, we measure the singular-value spectra of the full-rank reference coefficient matrices at all $R = 11$ residual stages, before RankVQ factorization. These matrices include A, B, A_v, and B_v. For each matrix at stage r, we compute

$$E_r(k) = \frac{\sum_{j=1}^{k} \sigma_{r,j}^2}{\sum_j \sigma_{r,j}^2}. \tag{40}$$

Table 13 reports the min–max spectral energy across the 11 stages. Spectral concentration is observed for all four coefficient matrices throughout the residual stages. Even the lowest Top-16 energy ratio is 72%, showing that the structure observed in Fig. 1 is not confined to the first residual stage.

Table 13: Spectral energy across the 11 residual stages under the default 1-bit configuration. Values report the min–max across stages.

Coefficient matrix	Top-4 (%)	Top-8 (%)	Top-16 (%)
A	51–73	61–85	79–98
B	52–74	63–87	81–97
A_v	46–69	57–81	72–94
B_v	46–69	56–81	73–92

F.8 Additional Ablations

Regularization coefficient λ. Table 14 reports RankVQ-1 performance on LongBench and InfiniteBench across a range of λ values. A very small λ allows the M-step to overfit noisy coefficient estimates $\bar{A}^\star, \bar{B}^\star$ and produces slightly less stable low-rank factors, while a very large λ biases the factors toward zero and underfits. The method is not sensitive to λ within roughly one order of magnitude around the default $\lambda = 10^{-3}$.

Table 14: Sensitivity to the regularization coefficient λ under 1-bit quantization on LLaMA-3.1-8B-Instruct.

λ	Avg. bit (↓)	LongBench (↑)	InfiniteBench (↑)
10^{-5}	0.86	44.92	36.19
10^{-4}	0.86	45.88	37.97
10^{-3} (default)	0.86	**46.36**	**39.42**
10^{-2}	0.86	45.71	38.74
10^{-1}	0.86	43.97	37.69

Factor initialization. Table 15 compares three initialization schemes for the low-rank factors: random Gaussian initialization [42], truncated SVD [43] based on the full-rank reference solution (our default), and identity-like initialization. The SVD-based initialization achieves the best final performance on both benchmarks. Random Gaussian and identity-like initialization also provide viable starting points, but both yield lower final performance than truncated SVD.

Table 15: Effect of low-rank factor initialization under 1-bit quantization on LLaMA-3.1-8B-Instruct.

Initialization	Avg. bit (↓)	LongBench (↑)	InfiniteBench (↑)
Random Gaussian	0.86	45.08	37.16
Identity-like	0.86	43.91	38.94
Truncated SVD (default, ours)	0.86	**46.36**	**39.42**

Full-rank reference variability. At the default $N_r = N_c = 32$, rank-32 SVD reconstructs a fixed full-rank reference up to numerical precision. We therefore examine the variability of independently constructed full-rank references. Across the originally reported run and ten additional runs, the LongBench score is 42.80 ± 0.48, with a range of $[42.11, 43.64]$. The reported B+SVD score of 43.25 lies within this range. Thus, the difference between the reported B and B+SVD results should not be attributed to SVD.

Calibration-draw robustness. Since the codebooks are constructed from sampled calibration tokens, we evaluate whether the 1-bit LongBench results are sensitive to the calibration subset. Table 16 reports the mean $\pm$ sample standard deviation over three independent calibration draws on LLaMA-3.1-8B-Instruct. All other hyperparameters are kept fixed, and each method uses its default 1-bit configuration. RankVQ-1 remains consistently stronger than CommVQ-1 while using a lower deployed Avg. bit, indicating that the improvement is not due to a favorable calibration sample. The paired RankVQ-1 gains across the three draws are 1.42, 1.47, and 1.33 points, giving a mean gain of 1.41 ± 0.07. These repeated results support the consistency of the main 1-bit LongBench comparison, rather than statistical significance for every individual result.

Table 16: Calibration-draw robustness on LongBench under the default 1-bit configurations.

Method	Avg. bit (↓)	Draw 1	Draw 2	Draw 3	Mean $\pm$ Std.
CommVQ-1	1.03	44.94	44.61	45.18	44.91 ± 0.29
RankVQ-1 (ours)	0.86	46.36	46.08	46.51	46.32 ± 0.22

Robustness Analysis. Since the codebook is trained on a subset of FineWeb-Edu [30], we further test its robustness under domain shift. Table 17 compares the perplexity of FP16, CommVQ-1, and RankVQ-1 on FineWeb-Edu, GSM8K [38], RepoBench-P [36], and KV_Retrieval [37]. RankVQ-1 introduces only limited perplexity increases over FP16, with PPL differences ranging from +0.13 to +1.15 across the four datasets. In contrast, CommVQ-1 suffers noticeably larger degradation, especially on FineWeb-Edu and RepoBench-P, where its PPL gaps reach +4.61 and +4.13, respectively. This consistent advantage indicates that the codebook constructed by RankVQ is more robust to domain shift.

Table 17: Perplexity (PPL) comparison across different domains. The last row reports the PPL increase over FP16 for CommVQ-1 / RankVQ-1.

Method	FineWeb-Edu	GSM8K	RepoBench-P	KV_Retrieval
FP16	10.17	5.67	2.20	31.93
CommVQ-1	14.78	7.06	6.33	34.17
RankVQ-1	11.32	6.07	2.54	32.06
PPL Diff vs. FP16 (↓)	+4.61/**+1.15**	+1.39/**+0.40**	+4.13/**+0.34**	+2.24/**+0.13**

F.9 Additional Efficiency Results

1-bit deployment efficiency. To complement the 128K result in Section 5.5, Table 18 provides additional measurements at 8K, 16K, and 32K contexts on LLaMA-3.1-8B-Instruct, including decoding latency, throughput, and deployed KV cache memory. All measurements use batch size 1 and are taken on a single NVIDIA H100-80GB GPU. FP16 cache memory grows linearly with context length, consistent with the 16.80 GB value reported at 128K in Table 6. RankVQ-1 consistently maintains the smallest deployed cache footprint among all compared methods, while its decoding latency and throughput remain competitive with existing quantization baselines at shorter contexts and become strictly better than CommVQ-1 [19] and KIVI-1 [26] at 32K. These trends are consistent with the main-text observation that reducing the deployed KV cache footprint yields increasing practical benefits in longer-context inference.

Table 18: 1-bit efficiency comparison with batch size 1 on LLaMA-3.1-8B-Instruct.

Context	Methods	Latency (ms/token) ↓	Throughput (token/s) ↑	KV cache (GB) ↓
8K	FP16	18.61	53.73	1.05
	KIVI-1	20.14	49.65	0.20
	CommVQ-1	21.73	46.02	0.15
	RankVQ-1	**19.60**	**51.02**	**0.09**
16K	FP16	26.82	37.29	2.14
	KIVI-1	29.67	33.70	0.36
	CommVQ-1	28.19	35.47	0.28
	RankVQ-1	**27.07**	**36.94**	**0.18**
32K	FP16	46.51	21.50	4.21
	KIVI-1	52.17	19.17	0.78
	CommVQ-1	49.36	20.26	0.56
	RankVQ-1	**44.92**	**22.26**	**0.34**

2-bit deployment efficiency. We next report deployment efficiency for the 2-bit configurations across context lengths from 8K to 128K. All measurements use LLaMA-3.1-8B-Instruct with batch size 1 on a single NVIDIA H100-80GB GPU, following the same profiling setup as the 1-bit evaluation. Table 19 shows that RankVQ-2 consistently uses the smallest deployed KV cache among the compared methods. At 128K, RankVQ-2 uses 2.38 GB and achieves 68.07 ms/token, compared with 3.51 GB and 75.36 ms/token for CommVQ-2.

Table 19: 2-bit efficiency comparison with batch size 1 on LLaMA-3.1-8B-Instruct.

Context	Method	Latency (ms/token) ↓	Throughput (token/s) ↑	KV cache (GB) ↓
8K	FP16	18.61	53.73	1.05
	KIVI-2	20.11	49.73	0.29
	CommVQ-2	19.07	52.44	0.27
	RankVQ-2	**18.73**	**53.39**	**0.18**
16K	FP16	26.82	37.29	2.14
	KIVI-2	30.08	33.24	0.50
	CommVQ-2	29.76	33.60	0.47
	RankVQ-2	**27.01**	**37.02**	**0.36**
32K	FP16	46.51	21.50	4.21
	KIVI-2	53.75	18.60	1.05
	CommVQ-2	51.91	19.26	0.98
	RankVQ-2	**48.22**	**20.74**	**0.62**
128K	FP16	63.78	15.68	16.80
	KIVI-2	76.91	13.00	4.07
	CommVQ-2	75.36	13.27	3.51
	RankVQ-2	**68.07**	**14.69**	**2.38**

Analytical bit-width versus measured memory. The analytical Avg. bit in Eq. 23 measures the idealized number of stored index bits per cached scalar. In contrast, the measured KV cache footprint

reported in Table 6 also includes implementation-level allocations such as packed or unpacked index tensors, deployed codebooks, runtime buffers, padding, and framework allocator overhead. To make this connection explicit, Table 20 decomposes the measured 128K memory footprint into the analytical payload and the remaining implementation overhead under the same profiling setup. The same trend holds in the 2-bit setting: at 128K, RankVQ-2 has an analytical payload of 1.72 GB and an implementation overhead of 0.66 GB, yielding a total measured footprint of 2.38 GB.

Table 20: Memory decomposition at 128K context length and batch size 1. The implementation overhead collects runtime buffers, padding, deployed codebooks, and framework allocator effects.

Method	Avg. bit	Analytical payload	Implementation overhead	Total measured
KIVI-1	2.00	2.10	0.88	2.98
CommVQ-1	1.03	1.08	1.37	2.45
RankVQ-1	0.86	0.90	0.54	1.44

G Limitations

RankVQ inherits the commutativity framework from CommVQ and therefore relies on RoPE-based architectures; extending the approach to other positional encodings is left for future work. Offline codebook construction requires a calibration set and incurs additional EM training cost, though this cost is amortized across all future inference runs. Our task-performance evaluations focus on 7–8B open-source models, leaving evaluation of task performance on substantially larger backbones for future work.

H Broader impact

As a compression method for long-context LLM inference, RankVQ primarily reduces GPU memory cost and can lower the hardware barrier for deploying long-context models, which is generally positive. At the same time, improved inference efficiency may also lower the cost of deploying capable LLM systems at scale, which could indirectly amplify existing misuse risks associated with such models. Our work does not release new models or datasets, and it does not differentially affect any specific user group.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Appendix G.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: See Appendix D.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See Section 5 and Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The code will be open-sourced upon publication.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See Section 5.1 and Appendix B.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Error bars are reported for the most calibration-sensitive 1-bit LongBench setting in Appendix F.8 (Table 16). The main-text tables report single-run point estimates.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix F.9 and Table 20.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We comply.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See Section H.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: Existing assets are cited, but their licenses are not explicitly listed in the paper. See Section 5.1.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We do not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: This paper does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.