VQKV: High-Fidelity and High-Ratio Cache Compression via Vector Quantization

Anonymous ACL submission

Abstract

The growing context length of Large Language Models (LLMs) enlarges the Key-Value (KV) cache, limiting deployment in resource-limited environments. Prior LLM-weight-training-free approaches for KV cache compression typically rely on low-rank approximation or scalar quantization, which fail to simultaneously achieve high KV-cache reduction ratios and high reconstruction fidelity. We propose VQKV, a novel, LLM-weight-training-free method introducing vector quantization (VQ) to obtain highly compressed KV representations while preserving high model fidelity, allowing for the representation of thousands of floating-point values with just a few integer indices. As a result, VQKV achieves an 82.8% KV-cache reduction ratio on LLaMA3.1-8B while retaining 98.8% of the baseline performance on LongBench and enabling more than $4.3\times$ longer generation under the same single-GPU memory budget.

1 Introduction

Large Language Models (LLMs) increasingly rely on long contexts, but the Key-Value (KV) cache grows linearly with sequence length and quickly limits deployment under fixed memory budgets. Reducing this footprint without modifying model weights is especially attractive because an LLM-weight-training-free method can be applied to existing checkpoints without retraining or changing model behavior.

Existing KV cache compression methods make different compromises. Training-based architectural methods such as MLA (Liu et al., 2024a) and CSKV (Wang et al., 2024) can provide strong compression, but require modifying or continuing to train the LLM weights. Training-free token-level or retrieval-based methods, including H2O (Zhang et al., 2023b), SnapKV (Li et al., 2024b), and PQCache (Zhang et al., 2025), shorten the effective sequence length, but discarded tokens cause irreversible information loss and brittle long-context recall. Token-preserving methods based on low-rank approximation, scalar precision reduction, or channel-wise coding, such as Palu (Chang et al., 2024), ASVD (Yuan et al., 2025), KIVI (Liu et al., 2024b), KVQuant (Hooper et al., 2024), and coupled quantization (CQ) (Zhang et al., 2024), avoid token eviction but degrade rapidly when pushed to very high KV-cache reduction ratios. This leaves a gap for a training-free method that preserves all tokens while directly compressing feature-dimensional KV vectors at very high ratios.

Vector quantization (VQ) (Gersho and Gray, 1991; van den Oord et al., 2017) maps continuous vectors to compact discrete indices and has been effective in speech representation learning (Baevski et al., 2020; Hsu et al., 2021; Zeghidour et al., 2021), visual tokenization (van den Oord et al., 2017; Esser et al., 2021; Zheng et al., 2022), and large-scale vector retrieval (Jegou et al., 2010; Douze et al., 2025; Johnson et al., 2019). Recent KV-cache VQ methods demonstrate this potential but remain limited: RVQ (Kumar, 2024) suffers from insufficient reconstruction fidelity; CommVQ improves efficiency by constraining quantization to a RoPE-commutative pre-RoPE path, but its indirect target may reduce fidelity to the post-RoPE key states used in attention, especially under aggressive compression and long contexts; PQCache (Zhang et al., 2025) combines product quantization with token retrieval, making it closer to retrieval-based token selection than a pure compression-and-reconstruction approach; and AnTKV (Li et al., 2025b) uses non-uniform allocation that can create GPU computation bubbles and reduce inference efficiency. Overall, maintaining high reconstruction fidelity at very high KV-cache reduction ratios remains difficult, especially because LLM outputs are highly sensitive to KV perturbations and post-RoPE key states are hard to reconstruct accurately.

These observations motivate VQKV, an LLM-weight-training-free KV cache compression framework that uniformly compresses KV vectors while preserving all tokens. VQKV learns RSimVQ codebooks offline and encodes each intermediate KV vector into a small set of discrete indices, directly targeting high-fidelity reconstruction of the post-RoPE cache. During decoding, a fused FlashAttention–VQ kernel performs codebook lookup, partial reconstruction, and attention computation on demand, avoiding full-cache materialization while keeping runtime overhead controlled. This design enables high KV-cache reduction ratios while maintaining strong downstream performance.

Extensive evaluations on LLaMA3.1-8B (Grattafiori et al., 2024) and LLaMA3.2-3B (Meta AI, 2024) across LongBench (Bai et al., 2024), NIAH (Li et al., 2024a), and RULER (Hsieh et al., 2024) demonstrate that our VQKV achieves an 82.8% KV-cache reduction ratio on LLaMA3.1-8B while retaining 98.8% of the full-cache LongBench performance. Remarkably, on some tasks, VQKV even surpasses the performance of the uncompressed full-cache baseline. Under the same 48 GB GPU setting, this memory efficiency extends the maximum generation length from approximately 190k tokens to over 824k tokens, yielding more than $4.3\times$ longer usable context than the original model.

2 Related Work

KV Cache Compression Existing KV cache compression methods mainly follow token eviction, feature-dimension reduction, or scalar quantization, each trading KV-cache reduction ratio against model fidelity. Token eviction methods (Xiao et al., 2024; Zhang et al., 2023b; Li et al., 2024b; Cai et al., 2025; Kai et al., 2026) reduce memory by discarding or organizing less critical tokens, but shorten the effective context and introduce irreversible information loss. Feature-dimension reduction methods (Liu et al., 2024a; Chang et al., 2024; Yuan et al., 2025) project KV vectors into lower-dimensional spaces through learned projections or low-rank decomposition; strong compression often requires additional LLM training, as in MLA (Liu et al., 2024a) and CSKV (Wang et al., 2024), or sacrifices reconstruction fidelity in training-free settings. Scalar quantization methods (Liu et al., 2024b; Hooper et al., 2024; Chang et al., 2024) reduce per-value precision and are easy to deploy, but treat features independently and thus miss vector-level correlations. Overall, token eviction hurts recall, feature reduction can require training or lose fidelity, and scalar quantization struggles at aggressive bit-widths, motivating a high-ratio method that preserves the information in each KV vector.

Vector Quantization Vector quantization is a widely recognized technique known for its effective compression and representation abilities. Wav2Vec (Baevski et al., 2020) and HuBERT (Hsu et al., 2021) use VQ to extract discrete pseudo labels from continuous raw waves for unsupervised learning. SoundStream (Zeghidour et al., 2021) introduces residual vector quantization, thus enriching the representation space of codebooks. In computer vision, VQ is generally used for concentrating information from pixels (He et al., 2022; van den Oord et al., 2017; Esser et al., 2021; Zheng et al., 2022). At the same time, VQ helps introduce information from other modalities into LLMs (Liu et al., 2025a; Zhang et al., 2023a). Other works use VQ for constraining the specialized formats of generation, such as code generation (Liu et al., 2025c) and action generation (Wang et al., 2025). Previous work has shown great compression and reconstruction capability of VQ for continuous vectors. Recent work has begun to explore VQ for KV cache compression, including RVQ-based KV compression (RVQ) (Kumar, 2024), CommVQ's RoPE-commutative vector quantization (Li et al., 2025a), PQCache's product-quantization-based retrieval (Zhang et al., 2025), and AnTKV's anchor-token-aware non-uniform quantization (Li et al., 2025b). Our VQKV instead uses RSimVQ to preserve reconstruction fidelity under substantially higher KV-cache reduction ratios, enabling uniform feature-dimensional KV-vector compression without token retrieval or anchor-token preservation.

3 Compressing KV with Vector Quantization

In this section, we introduce how VQKV employs vector quantization to compress the KV cache to a high KV-cache reduction ratio while enabling high-fidelity reconstruction. We first describe how VQ codebooks are learned, followed by how VQKV is integrated into both prefilling and decoding. VQKV follows a three-stage pipeline: it first trains RSimVQ codebooks offline from cached KV vec-

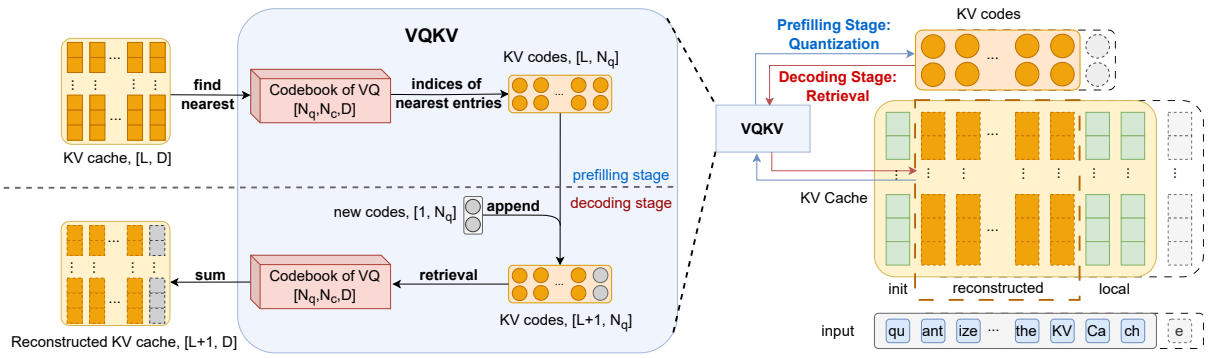


Figure 1: Overview of our VQKV. The left part shows the detailed process of our VQKV on prefilling stage and decoding stage. The right part shows the overview of our VQKV.

tors, then uses the learned codebooks to compress the intermediate cache into discrete indices, and finally reconstructs the required cache blocks on demand during decoding. Figure 1 illustrates the overall workflow of VQKV, where the **left** side depicts the compression–reconstruction process of the KV cache, and the **right** side presents how VQKV is integrated into the prefill and decoding stages. During **prefill**, VQKV compresses each KV cache vector by mapping it to the nearest entries in multiple codebooks and storing only the corresponding indices as KV codes. During **decoding**, VQKV compresses the cache of each new token and updates the stored KV codes while maintaining a *local* sliding window by discarding the oldest entries. When generating new tokens, VQKV performs an on-demand reconstruction of the KV cache from the codebooks.

3.1 Learning High-fidelity VQ Codebooks

We use Residual Simple Vector Quantization (RSimVQ) to compress each KV vector. Combining Residual VQ (Zeghidour et al., 2021) with SimVQ (Zhu et al., 2025), RSimVQ consists of multiple residual-connected codebooks and has an extra projection matrix on every codebook. During compression, each codebook represents an independent subspace of the original cache with the nearest codebook-entry index. During reconstruction, RSimVQ retrieves entries from the codebooks and sums them to reconstruct the original cache. As shown in Figure 2, RSimVQ provides a better reconstruction–parameter trade-off than other VQ variants on LLaMA3.1-8B key-cache vectors, motivating its use as the quantizer in VQKV.

Formally, let $x \in \mathbb{R}^D$ be one key/value vector from the KV cache and $\hat{x}$ be its reconstruction. Let Q_i denote a codebook with entries $q_1, \ldots, q_S$,

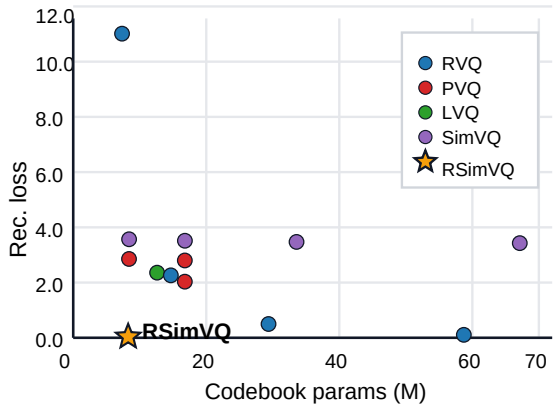


Figure 2: Comparison of VQ variants for LLaMA3.1-8B key-cache reconstruction. RSimVQ, used by VQKV, achieves substantially lower evaluation reconstruction loss with a compact VQ model size.

where S is the codebook size. For each codebook, RSimVQ first finds the nearest entry q_z and records the corresponding index z.

$$z = \underset{j \in \{1,\ldots,S\}}{\operatorname{argmin}} \|x - Wq_j\|, \quad \hat{x} \triangleq Wq_z \tag{1}$$

where W is the projection matrix of each codebook and z is the selected index. Then, RSimVQ iteratively sends the residual part $(x - \hat{x})$ to Q_{i+1} for next step quantization.

$$x \leftarrow x - \hat{x}, \quad z_i \leftarrow z \tag{2}$$

After N iterations, RSimVQ maps the original cache vector into a small set of integer codes $z_1, \ldots, z_N$. VQKV stores only the codebooks and quantized codes in memory instead of the original cache vectors, reducing the original KV-cache representation from $[L, D]$ floats (ignoring layer and head dimensions for simplicity) to $[L, N]$ integers. When reconstructing, RSimVQ simply looks

up entries according to these codes and sums the entries.

$$\hat{x} = \sum_{i=1}^{N} W_i Q_i[z_i] \tag{3}$$

We collect KV caches from approximately 10M tokens to train these codebooks, with separate codebook sets for keys and values. During training, RSimVQ uses a stop-gradient operation $\mathrm{sg}(\cdot)$ to keep gradient flowing after discretization, along with the training loss as follows:

$$\mathcal{L} = \|x-\hat{x}\|^2+\beta\|Wq_z-\mathrm{sg}(x)\|^2+\gamma\|x-\mathrm{sg}(Wq_z)\|^2 \tag{4}$$

3.2 Prefilling Stage

The codebooks used in VQKV are trained independently and can be directly integrated without modifying or fine-tuning the original LLM parameters, making VQKV LLM-weight-training-free. Therefore, the model's performance is only minimally affected, as long as the compression algorithm can reconstruct the KV cache with sufficient fidelity.

We preserve both an *initial* segment of the KV cache of length L_{init} and the most recent segment of length L_{local} from compression (seen in the right part of Figure 1). For simplicity, the following discussion considers only the *intermediate* KV cache to be compressed.

As discussed above, we use two sets of codebooks to compress *intermediate* cache K^m, V^m into KV codes K^c, V^c correspondingly. Take the key cache compression as an example. Let the codebook of the key RSimVQ be Q_i^k, and suppose that the codebook consists of S_k independent entries $q_1, q_2, \ldots, q_{S_k}$. When prefilling, every cache vector $k \in K^m$ finds the nearest entry in the first codebook Q_1^k and obtains the corresponding index z_1^k.

$$z_1^k = \underset{j\in\{1,\ldots,S^k\}}{\operatorname{argmin}} \|k - W_1^k q_j\| \tag{5}$$

Then, the residual part $(k - W_1^k q_{z_1^k})$ is sent to codebook Q_2^k for next step quantization.

$$k \leftarrow k - W_1^k q_{z_1^k} \tag{6}$$

After N^k iterations, the original vector k can be represented as N^k integers $(z_1^k, z_2^k, \ldots, z_{N^k}^k)$, each no greater than S^k. As a result, vectors that originally had a dimensionality of $\mathbb{R}^{D^k}$ can now be represented using only N^k integers, with each integer occupying only $\log S^k$ bits.

The whole compression process can be expressed iteratively as follows.

$$z_i^k = \underset{j\in\{1,\ldots,S^k\}}{\operatorname{argmin}} \|k_i - W_i^k q_j\|$$
$$k_1 = k, \quad k_{i+1} = k_i - W_i^k q_{z_i^k}, \quad i = 1, \ldots, N^k \tag{7}$$

$$z^k = \mathcal{VQ}^k(k) = (z_1^k, z_2^k, \ldots, z_{N^k}^k) \tag{8}$$

This procedure is applied to every key/value vector, compressing the original floating-point representations of size $K^m \in \mathbb{R}^{L\times D^k}$ into a small number of integer indices $K^c \in \mathbb{N}^{L\times N^k}$, significantly reducing the memory cost of KV-cache storage. After compression, we use the **original** KV cache for forward propagation during prefill.

$$O = \mathrm{attention}(Q, K, V) \tag{9}$$

3.3 Decoding Stage

In the decoding stage, we first compress the cache that moves out of the *local* segment and append the resulting codes to the previous compressed KV codes K^c. Then VQKV reconstructs the *intermediate* KV cache by selecting the entries from each codebook according to the stored indices. We continue to use the key cache as an illustrative example. For each stored key-index tuple $z^k \in K^c$, the reconstructed key vector $\hat{k}$ is calculated by summing the corresponding entries in the codebooks.

$$\hat{k} = \mathcal{VQ}^k[z^k] = \sum_{i=1}^{N^k} W_i^k Q_i^k[z_i^k] \tag{10}$$

With FlashAttention (Dao et al., 2022; Dao, 2024), it is unnecessary to reconstruct the full KV cache for the entire sequence at each decoding step. Instead, when reconstructing the cache, we only rebuild the portion required for the current decoding step, aligned with the FlashAttention computation flow, thereby further reducing the memory footprint of VQKV.

$$\hat{k}_m = \mathcal{VQ}^k[z_m^k], \quad \hat{v}_m = \mathcal{VQ}^v[z_m^v] \tag{11}$$
$$o_m = \mathrm{flash_attention}(q_{t+1}, \hat{k}_m, \hat{v}_m) \tag{12}$$

We implement a fused FlashAttention–VQ kernel that combines codebook lookup, partial reconstruction, and attention computation, avoiding full-sequence reconstruction before attention. The kernel further reduces overhead with three optimizations: a paged KV cache for the uncompressed

	Ratio	Single-Doc			Multi-Doc			Summary			Few-shot			Synthetic		Code		Avg.
		NQ	Qsp	MF	HQ	WQ	Msq	GR	QS	MN	TR	TQ	SS	PC	PR	LCC	Re-P	
LLaMA3.1-8B	0	13.0	20.4	32.1	12.0	14.0	8.7	29.7	25.2	0.9	73.5	91.0	47.2	0.8	26.8	72.0	69.2	33.5
+ SnapKV	80.0%	12.7	19.5	31.2	11.5	13.9	8.9	23.7	24.1	0.9	70.0	91.3	**46.5**	0.9	27.8	71.7	68.5	31.7
+ KIVI	82.8%	13.9	19.6	30.7	11.8	12.8	8.1	25.3	**24.9**	0.8	73.5	**91.4**	45.3	0.6	27.3	70.4	67.4	32.7
+ ASVD	80.0%	0.5	3.2	2.4	1.0	1.2	1.2	4.3	5.4	4.3	12.5	3.0	5.4	**3.1**	4.1	23.7	27.4	6.4
+ Palu	80.0%	6.7	18.2	22.7	11.7	11.9	6.5	24.6	23.6	5.6	73.5	87.8	44.8	2.4	5.1	56.6	54.0	28.5
+ RVQ	80.0%	1.7	8.7	13.2	4.6	6.8	3.1	20.0	13.5	**18.3**	2.0	25.1	14.5	0.1	0.5	31.7	38.2	12.6
+ CommVQ	82.8%	9.6	19.6	**36.1**	**13.2**	**16.5**	**10.3**	23.7	22.3	1.0	70.5	91.3	40.3	0.5	**37.0**	65.1	61.0	32.4
+ CQ	82.8%	9.6	19.0	21.5	9.5	10.6	7.3	**28.8**	24.7	0.6	**74.0**	90.1	43.9	2.3	10.0	62.9	59.7	29.7
+ **VQKV**	82.8%	**14.5**	**19.8**	30.5	11.6	14.0	8.2	26.2	23.9	0.8	73.5	90.5	45.8	0.8	28.2	**71.8**	**69.4**	**33.1**
LLaMA3.2-3B	0	10.3	21.6	35.5	9.5	12.8	6.8	30.1	23.8	28.3	69.5	87.2	38.3	0.0	7.0	70.0	66.1	32.3
+ SnapKV	80.0%	10.5	21.6	34.1	9.4	12.6	7.2	23.1	**24.1**	27.1	65.0	87.2	38.4	0.0	7.0	69.8	65.1	31.4
+ KIVI	82.8%	7.4	**22.1**	33.1	9.8	12.0	5.2	17.4	14.2	16.0	**69.5**	**88.3**	30.3	1.2	**7.4**	70.0	**65.6**	29.3
+ ASVD	80.0%	0.8	10.7	8.8	5.4	5.2	2.7	8.7	8.6	11.1	31.5	49.1	17.7	**3.3**	3.5	35.4	36.6	14.9
+ Palu	80.0%	**12.2**	16.7	27.7	9.9	11.6	6.6	24.2	20.0	19.4	58.0	86.4	**41.4**	0.3	4.7	56.2	54.3	28.1
+ RVQ	80.0%	0.7	3.0	6.5	1.3	3.2	1.4	9.5	5.3	12.2	1.0	7.5	8.3	0.9	0.3	28.5	30.8	7.5
+ CommVQ	82.4%	5.5	20.3	34.8	**10.1**	12.9	**7.6**	28.3	23.2	24.9	**69.5**	87.1	38.1	0.0	**7.4**	67.8	62.1	31.2
+ CQ	82.4%	4.6	19.3	22.5	8.5	10.5	7.1	25.9	22.4	23.8	68.0	84.7	37.8	0.0	4.2	64.5	62.4	29.2
+ **VQKV**	82.4%	10.7	20.6	**36.7**	9.5	**14.5**	6.6	**29.0**	23.1	**27.5**	**69.5**	87.2	38.5	0.0	7.0	**70.6**	64.4	**32.2**

Table 1: Results of LLaMA3.1-8B and LLaMA3.2-3B on LongBench. The **Ratio** column reports KV-cache reduction ratios.

local window, query-codebook precomputation for compressed keys, and fused value lookup in the weighted-sum path. For compressed keys, the kernel first precomputes products between the current query and each key codebook entry:

$$P_i^k[s] = q_{t+1}^\top W_i^k Q_i^k[s], \quad s = 1, \dots, S^k. \quad (13)$$

Then each compressed-token attention score is obtained by summing indexed entries,

$$q_{t+1}^\top \hat{k}_m = \sum_{i=1}^{N^k} P_i^k[z_{m,i}^k], \quad (14)$$

avoiding repeated materialization of $\hat{k}_m$ and repeated query–key dot products. For values, lookup and accumulation are fused directly into the weighted sum instead of writing a reconstructed value cache to memory.

We also optimize online quantization with blockwise distance computation and batched processing over the local segment. Instead of quantizing one token at every decoding step, VQKV quantizes the whole outgoing local segment K^l, V^l together, reducing peak distance-computation memory and performing VQ compression once every L_{local} steps.

3.4 KV-cache Reduction Ratio

The total KV-cache reduction ratio of VQKV is determined by the number and size of codebooks. For each vector in the KV cache, the original representation consists of D 16-bit floating-point values, where D denotes the head dimension. After compression, each vector is represented using only N integers, where each integer indexes a codebook of size S. Consequently, the compressed representation requires only $N \times \log S$ bits. For KV cache with dimensions of D^k and D^v, our VQKV achieves a KV-cache reduction ratio r such that

$$r = \left(1 - \frac{N^k \log S^k + N^v \log S^v}{16(D^k + D^v)}\right) \times 100\% \quad (15)$$

4 Experiment

In this section, we evaluate VQKV across different models and downstream tasks. We first compare the main VQKV configuration with feature-reduction, scalar/channel quantization, vector-quantization, and token-retention baselines on long-context benchmarks. The main comparison uses the pure VQKV setting at 82.8% compression on LLaMA3.1-8B and 82.4% compression on LLaMA3.2-3B. We then evaluate high-ratio compression, memory usage, maximum generation length, decoding time, and ablation settings to understand where the compression gains come from.

4.1 Setup

We conduct all our experiments on LLaMA3.1-8B (Grattafiori et al., 2024) and LLaMA3.2-3B (Meta

AI, 2024). For both models, we set the initial segment length L_{init} to 4 and the local segment length L_{local} to 1024. We randomly sample 0.1% of OpenWebText (Gokaslan and Cohen, 2019) for RSimVQ training, and each RSimVQ model is trained with a learning rate of 0.001 and a batch size of 65536. The codebook sizes vary across cache types and models. For LLaMA3.1-8B (Grattafiori et al., 2024), we use $(N^k, N^v) = (56, 16)$ codebooks with codebook sizes $(S^k, S^v) = (1024, 512)$, while for LLaMA3.2-3B (Meta AI, 2024), we use $(N^k, N^v) = (56, 10)$ codebooks and set the codebook sizes to $(S^k, S^v) = (1024, 65536)$. By Equation 15, our method achieves a KV-cache reduction ratio of 82.8% on LLaMA3.1-8B and 82.4% on LLaMA3.2-3B. All long-context evaluations use FP16 and FlashAttention. We compare against ASVD (Yuan et al., 2025), Palu (Chang et al., 2024), KIVI (Liu et al., 2024b), SnapKV (Li et al., 2024b), RVQ (Kumar, 2024), CommVQ (Li et al., 2025a), and CQ (Zhang et al., 2024). For the main 82.8%/82.4% comparison, we align each baseline to the target KV-cache reduction ratio using the closest available configuration and report its measured ratio in the **Ratio** column. SnapKV keeps 2048 recalled middle tokens, KIVI uses its supported bit-width setting, ASVD uses ratio parameter 0.2, Palu retains 70% of the KV cache, RVQ uses a codebook of shape $(32, 5, 1024, 16)$, CommVQ uses 22 key residuals and 6.4 total KV bits, and CQ uses a variant of 4c11b. All reported KV-cache reduction ratios follow the definitions used by the corresponding methods, and the measured ratios are listed in the **Ratio** column.

4.2 Long-Context Evaluation

The long-context evaluation experiments are all conducted on multiple NVIDIA A800 GPUs with FP16 precision. We evaluate our VQKV against other KV cache optimization methods with three long-context benchmarks on OpenCompass (Contributors, 2023): LongBench (Bai et al., 2024), Needle-In-A-Haystack (NIAH) (Li et al., 2024a), and RULER (Hsieh et al., 2024). All tasks use a truncation context length of 32K. The results of SnapKV on NIAH are taken from Liu et al. (2025b). Detailed NIAH heatmaps are provided in Appendix B; Table 2 reports the corresponding NIAH scores in the main text.

Across LongBench, NIAH, and RULER, VQKV provides the strongest overall compressed-cache result at the main 82.8% compression setting on LLaMA3.1-8B. It stays close to the full-cache LongBench score on both models, preserves exact NIAH retrieval, and leads the compressed baselines on RULER across all tested context lengths. These results show that VQKV preserves model capability under high compression not only on aggregate long-context tasks, but also on retrieval-style and length-controlled evaluations where KV reconstruction errors are more exposed.

	Ratio	NIAH	RULER				
			4K	8K	16K	32K	Avg.
LLaMA3.1-8B	0	100.0	94.7	92.7	93.5	90.2	92.8
+ SnapKV	80.0%	88.4	91.2	77.1	69.8	58.5	74.2
+ KIVI	82.8%	99.4	89.0	85.9	82.9	75.7	83.4
+ ASVD	80.0%	3.3	0.9	0.9	0.2	0.2	0.6
+ Palu	80.0%	69.0	89.9	84.7	83.4	78.0	84.0
+ RVQ	80.0%	39.6	9.2	4.9	4.2	1.5	5.0
+ CommVQ	82.8%	97.8	76.7	70.3	62.4	53.1	65.6
+ CQ	82.8%	99.7	91.4	87.4	81.6	72.2	83.2
+ VQKV	82.8%	**100.0**	**92.7**	**89.5**	**86.9**	**79.9**	**87.3**
LLaMA3.2-3B	0	100.0	90.0	85.9	83.3	78.3	84.4
+ SnapKV	80.0%	90.3	85.3	69.6	61.7	58.7	68.8
+ KIVI	75.0%	98.7	37.9	49.4	41.4	36.3	41.3
+ ASVD	80.0%	36.4	27.2	20.8	16.0	9.5	18.4
+ Palu	80.0%	36.0	71.8	65.4	59.7	55.1	63.0
+ RVQ	80.0%	25.1	4.0	2.3	1.3	1.1	2.2
+ CommVQ	82.4%	99.9	80.6	75.0	67.6	63.3	71.6
+ CQ	82.4%	97.7	79.1	68.1	58.6	51.2	64.2
+ VQKV	82.4%	**100.0**	**88.2**	**78.7**	**73.6**	**67.9**	**77.1**

Table 2: Results of LLaMA3.1-8B and LLaMA3.2-3B on NIAH and RULER. The **Ratio** column reports KV-cache reduction ratios.

As an additional compatibility check outside the main tables, combining VQKV with SnapKV reaches 96.6% compression on LLaMA3.1-8B and 96.5% on LLaMA3.2-3B while retaining 32.8 and 31.4 LongBench average scores, respectively, showing that feature-dimensional VQKV compression can be paired with sequence-dimension compression to achieve extremely high reduction ratios.

4.3 High-Ratio Compression Stress Test

To evaluate more aggressive compression, we sweep KV-cache reduction ratios on LLaMA3.1-8B and tune the controllable compression hyperparameters of ASVD, Palu, RVQ, CommVQ, CQ, and VQKV to match each target ratio as closely as possible. Figures 3, 4, and 5 summarize the sweeps, with the detailed results reported in Appendix C. At 90.0% compression, VQKV still preserves most of the full-cache capability and avoids the sharp

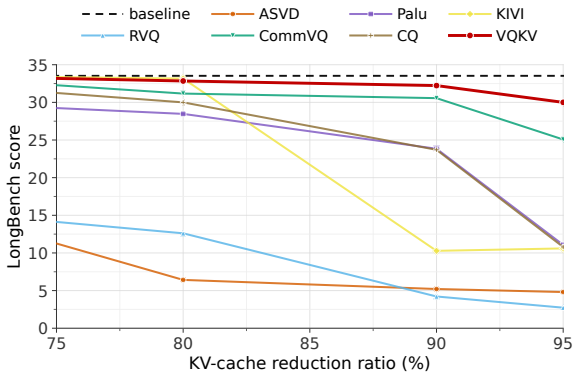


Figure 3: LongBench results of LLaMA3.1-8B under different KV-cache reduction ratios. VQKV remains stable at high KV-cache reduction ratios and degrades more gradually than competing methods.

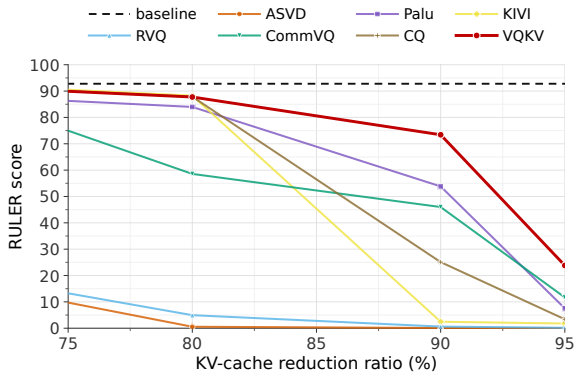


Figure 5: RULER results of LLaMA3.1-8B under different KV-cache reduction ratios. VQKV preserves the strongest long-context accuracy at 90.0% and 95.0% compression.

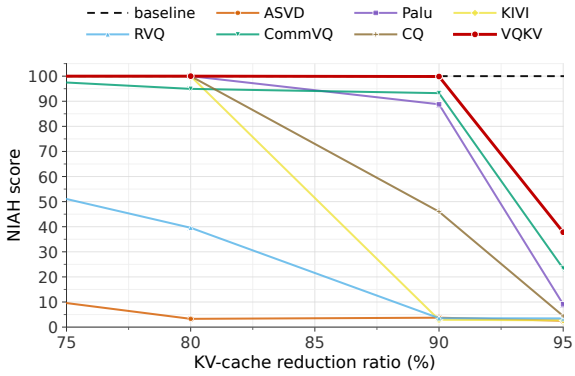


Figure 4: NIAH results of LLaMA3.1-8B under different KV-cache reduction ratios. VQKV maintains retrieval accuracy best under aggressive compression.

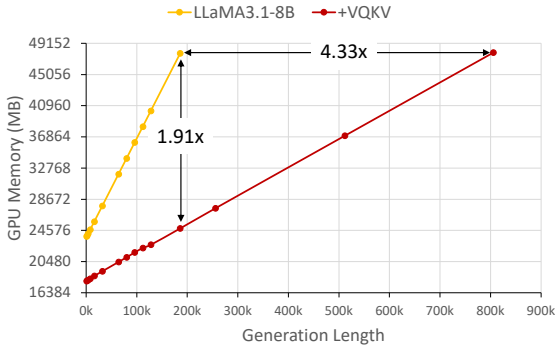


Figure 6: Generation length and corresponding peak memory usage between full-cache LLaMA3.1-8B and VQKV. The prompt length is 65536, and FlashAttention is used for both settings.

degradation observed in competing baselines. Even at the more aggressive 95.0% setting, where all methods become substantially harder to stabilize, VQKV remains the most robust method across the evaluated benchmarks. These results show that VQKV's main advantage is not only preserving performance at the 82.8% main setting, but also degrading more gracefully as the KV-cache reduction ratio becomes extreme.

4.4 Memory Efficiency

To show the memory efficiency of the main VQKV setting, we test the maximum generation length of LLaMA3.1-8B on a single NVIDIA RTX 6000 Ada (48 GB) GPU with FP16 and FlashAttention. As shown in Figure 6, our method consistently exhibits substantially lower peak memory usage than the baseline across different generation lengths. When the full-cache baseline model reaches its maximum generation length, VQKV consumes nearly half of the memory required by the baseline. More importantly, our approach dramatically extends the maximum achievable generation length: while the baseline LLaMA3.1-8B model encounters an out-of-memory error at approximately 190k tokens, our method supports sequences exceeding 824k tokens on a single NVIDIA RTX 6000 Ada (48 GB) GPU, representing a more than $4.3\times$ increase in usable context length. These results connect the 82.8% KV-cache reduction ratio to a practical long-context benefit: under the same memory budget, VQKV enables substantially longer generation without changing the model weights.

4.5 Time Efficiency

We further evaluate decoding time on LLaMA3.1-8B using NVIDIA A800 GPUs with FP16 precision and FlashAttention. For each method, we fix the batch size to 8, decode 256 new tokens, and vary the prompt length to measure total decoding time.

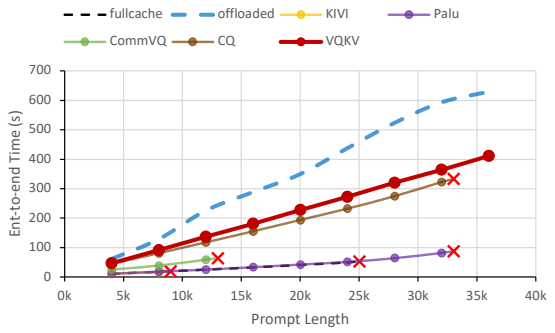


Figure 7: End-to-end decoding time for generating 256 tokens with LLaMA3.1-8B under different prompt lengths. All methods use FP16 FlashAttention with batch size 8, and red crosses denote out-of-memory cases.

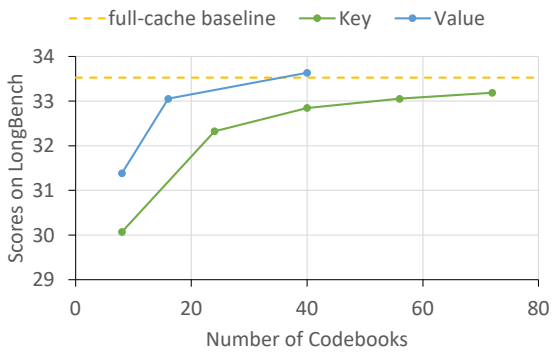


Figure 8: Results of VQKV on LongBench with different numbers of codebooks on LLaMA3.1-8B.

As shown in Figure 7, VQKV provides a favorable trade-off between decoding speed and long-prompt applicability under the measured prompt lengths. Red crosses indicate out-of-memory cases at the corresponding prompt lengths. In contrast, methods that can continue into the same long-prompt regime as VQKV incur higher end-to-end decoding time. Thus, VQKV is not the fastest at short contexts, but it offers a better balance between practical long-context support and decoding efficiency among the compared methods.

4.6 Ablation Study

We evaluate our VQKV on different combinations of codebook numbers and sizes with LLaMA3.1-8B (Grattafiori et al., 2024). On LongBench (Bai et al., 2024), NIAH (Li et al., 2024a), and RULER (Hsieh et al., 2024), the ablation results show how codebook capacity trades off against KV-cache reduction ratio. The detailed results are available in Appendix D.

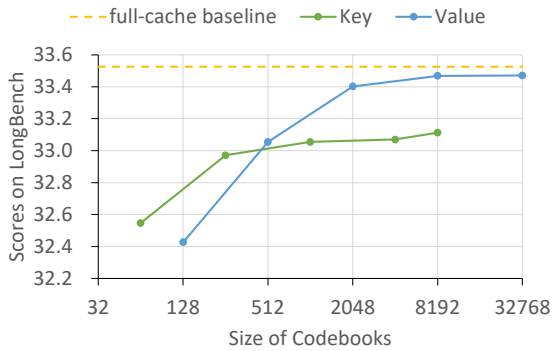


Figure 9: Results of VQKV on LongBench with different codebook sizes on LLaMA3.1-8B.

Figures 8 and 9 show that increasing codebook capacity generally improves performance, but also reduces the KV-cache reduction ratio. Larger codebooks and more codebooks both increase reconstruction capacity, with value-cache capacity having the clearest impact on long-context accuracy. The trade-off is direct: adding capacity improves fidelity but lowers the KV-cache reduction ratio, while overly small codebooks retain memory savings at the cost of noticeable accuracy loss.

We therefore use $(N^k, N^v) = (56, 16)$ and $(S^k, S^v) = (1024, 512)$ as the main LLaMA3.1-8B configuration. This setting provides a strong balance between KV-cache reduction ratio and reconstruction fidelity, and is used as the default configuration throughout the main experiments. Larger codebooks can further improve some metrics, but they move away from the high-compression regime that is the focus of this work.

5 Conclusion

In this paper, we proposed **VQKV**, an LLM-weight-training-free KV cache compression framework that leverages vector quantization to jointly capture correlations within cache vectors. By replacing continuous representations with compact discrete codes, VQKV achieves high KV-cache reduction ratios while preserving reconstruction fidelity. Extensive experiments on LLaMA3.1-8B and LLaMA3.2-3B show that VQKV consistently outperforms existing training-free methods and even surpasses the full-cache baseline in some cases. Moreover, VQKV achieves more than $4.3\times$ longer generation under the same single-GPU memory budget, demonstrating its effectiveness for memory-efficient long-context inference.

6 Limitations

Our method still has room for improvement in decoding efficiency. Although the customized Triton kernels substantially improve the runtime efficiency of VQKV, further work is needed to reduce the additional computational overhead introduced by VQ. We leave these optimizations for future work.

References

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2025. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. 2024. Palu: Compressing KV-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*.

OpenCompass Contributors. 2023. OpenCompass: A universal evaluation platform for foundation models.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The FAISS library. *IEEE Transactions on Big Data*.

Patrick Esser, Robin Rombach, and Bjorn Ommer. 2021. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883.

Allen Gersho and Robert M Gray. 1991. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media.

Aaron Gokaslan and Vanya Cohen. 2019. OpenWebText corpus.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009.

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

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. RULER: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. HuBERT: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM transactions on audio, speech, and language processing*, 29:3451–3460.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Jushi Kai, Yixuan Wang, Boyi Zeng, Haoli Bai, Bo Jiang, Ziwei He, and Zhouhan Lin. 2026. FreqKV: Key-value compression in frequency domain for context window extension. *arXiv preprint arXiv:2505.00570*.

Ankur Kumar. 2024. 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.

Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Colorado Reed, Chong

Wang, and Chuang Gan. 2025a. CommVQ: Commutative vector quantization for KV cache compression. *arXiv preprint arXiv:2506.18879*.

Mo Li, Songyang Zhang, Taolin Zhang, Haodong Duan, Yunxin Liu, and Kai Chen. 2024a. NeedleBench: Evaluating LLM retrieval and reasoning across varying information densities. *arXiv preprint arXiv:2407.11963*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024b. SnapKV: LLM knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Zeyu Li, Chuanfu Xiao, Yang Wang, Xiang Liu, Zhenheng Tang, Baotong Lu, Mao Yang, Xinyu Chen, and Xiaowen Chu. 2025b. AnTKV: Anchor token-aware sub-bit vector quantization for KV cache in large language models. *arXiv preprint arXiv:2506.19505*.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, and 1 others. 2024a. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Dongyang Liu, Shitian Zhao, Le Zhuo, Weifeng Lin, Yi Xin, Xinyue Li, Qi Qin, Yu Qiao, Hongsheng Li, and Peng Gao. 2025a. Lumina-mGPT: Illuminate flexible photorealistic text-to-image generation with multimodal generative pretraining. *arXiv preprint arXiv:2408.02657*.

Xiaoran Liu, Siyang He, Qiqi Wang, Ruixiao Li, Yuerong Song, Zhigeng Liu, Linlin Li, Qun Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025b. Beyond homogeneous attention: Memory-efficient LLMs via fourier-approximated KV cache. *arXiv preprint arXiv:2506.11886*.

Zihan Liu, Xinhao Luo, Junxian Guo, Wentao Ni, Yangjie Zhou, Yue Guan, Cong Guo, Weihao Cui, Yu Feng, Minyi Guo, and 1 others. 2025c. VQ-LLM: High-performance code generation for vector quantization augmented LLM inference. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1496–1509. IEEE.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024b. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. *arXiv preprint arXiv:2402.02750*.

Meta AI. 2024. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Luning Wang, Shiyao Li, Xuefei Ning, Zhihang Yuan, Shengen Yan, Guohao Dai, and Yu Wang. 2024. CSKV: Training-efficient channel shrinking for KV cache in long-context scenarios. *arXiv preprint arXiv:2409.10593*.

Yating Wang, Haoyi Zhu, Mingyu Liu, Jiange Yang, Hao-Shu Fang, and Tong He. 2025. VQ-VLA: Improving vision-language-action models via scaling vector-quantized action tokenizers. *arXiv preprint arXiv:2507.01016*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. 2025. ASVD: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*.

Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. 2021. SoundStream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:495–507.

Dong Zhang, Shimin Li, Xin Zhang, Jun Zhan, Pengyu Wang, Yaqian Zhou, and Xipeng Qiu. 2023a. SpeechGPT: Empowering large language models with intrinsic cross-modal conversational abilities. *arXiv preprint arXiv:2305.11000*.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. 2025. PQCache: Product quantization-based KVCache for long context LLM inference. *arXiv preprint arXiv:2407.12820*.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. 2024. KV cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *arXiv preprint arXiv:2405.03917*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023b. H2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Chuanxia Zheng, Tung-Long Vuong, Jianfei Cai, and Dinh Phung. 2022. MoVQ: Modulating quantized vectors for high-fidelity image generation. *Advances in Neural Information Processing Systems*, 35:23412–23425.

Yongxin Zhu, Bocheng Li, Yifei Xin, Zhihua Xia, and Linli Xu. 2025. Addressing representation collapse in vector quantized models with one linear layer. *arXiv preprint arXiv:2411.02038*.

A Responsible NLP and Reproducibility Details

A.1 Scientific Artifacts

Our study uses and builds upon existing scientific artifacts, including pretrained language models, benchmark datasets, evaluation scripts, and software libraries. Specifically, our experiments use publicly available large language models such as LLaMA3.1-8B and LLaMA3.2-3B, benchmark datasets and evaluation suites such as LongBench, NIAH and RULER, and open-source software packages including PyTorch, Transformers, FlashAttention and Triton. We cite the original papers, repositories, or model cards associated with these artifacts in the main text and bibliography.

Our work also creates scientific artifacts in the form of the VQKV implementation, VQKV Triton kernels, and evaluation scripts, which are intended to support reproducibility and further research on efficient long-context language model inference. We will release these artifacts after review.

A.2 Licenses and Terms of Use

We use existing artifacts in accordance with their respective licenses and terms of use. For pretrained models, we follow the licenses and use restrictions specified by the corresponding model providers. For benchmark datasets and evaluation suites, we follow the licenses or usage terms provided by the original dataset creators. For software libraries and baseline implementations, we follow their open-source licenses and cite the corresponding repositories or papers.

The artifacts created by this work, including code and checkpoints, are intended to be released under MIT. If some components depend on third-party artifacts with more restrictive terms, our release will respect the corresponding original access conditions and will not redistribute restricted model weights or datasets.

A.3 Intended Use of Artifacts

The existing artifacts used in this paper are used for their intended research purposes: pretrained language models are used for language modeling and downstream evaluation; benchmark datasets are used only for evaluating long-context understanding and generation; and software packages are used for model inference, quantization, and evaluation. We do not use these artifacts for deployment in user-facing systems, decision-making about individuals, surveillance, or other high-stakes applications.

The artifacts created by this work are intended for research on efficient long-context inference and KV-cache compression. They are not intended to be used to generate harmful content, to bypass access restrictions of proprietary models or datasets, or to support applications outside the access conditions of the original models and datasets.

A.4 Documentation and Data Statistics

We document the main artifacts used in our experiments in the experimental setup and implementation details. The evaluated models are listed in Section 4.1, the benchmark datasets and evaluation tasks are described in Section 4.1, and the implementation details are described in Section 4.1. Dataset statistics, including the number of tasks, context lengths, evaluation splits, and metrics, follow the official benchmark settings of LongBench, NIAH and RULER. We report the relevant dataset statistics and evaluation settings in Section 4.1.

A.5 Computational Experiments

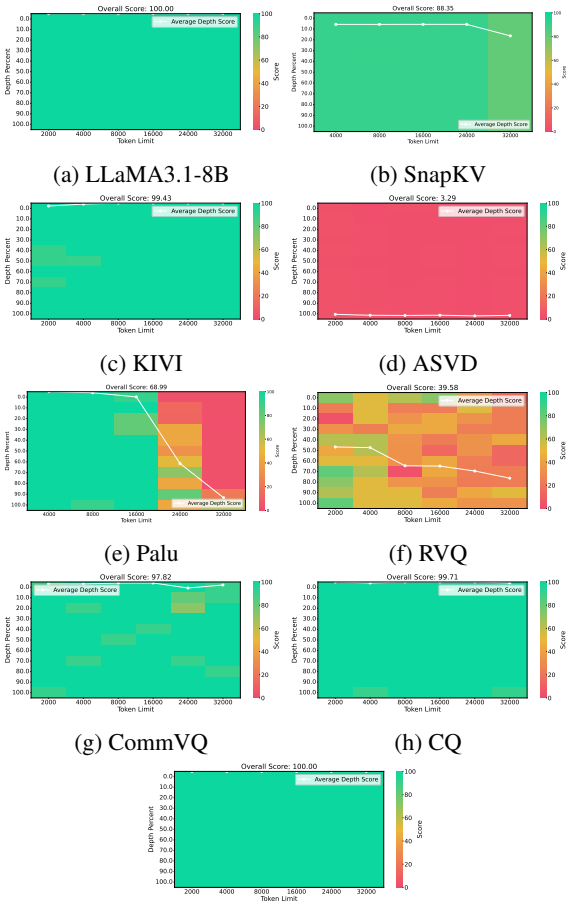
All reported results are obtained through computational experiments. We report the model sizes, evaluated context lengths, compression configurations, and hardware infrastructure in Section 4.1 and Section 4.4. Our experiments are conducted on NVIDIA A800 GPUs and a single NVIDIA RTX 6000 Ada GPU for the maximum-generation-length test.

We report the main hyperparameters and implementation settings, including codebook size, number of residual codebooks, quantization configuration, local window size, decoding setup, batch size, precision, and evaluation protocol, in Section 4.1 and Appendix D. Unless otherwise stated, we use the official evaluation scripts and default metrics of the corresponding benchmarks.

For the efficiency results, we report averages over multiple runs as specified in the corresponding tables. When multiple runs are conducted, we report summary statistics such as mean and standard deviation. When only a single deterministic evaluation run is used, we explicitly state this in the corresponding result table or caption.

A.6 Human Subjects

This work does not involve human subjects, human annotators, crowdworkers, or newly collected human-participant data. Therefore, questions related to participant instructions, recruitment, pay-



(a) LLaMA3.1-8B (b) SnapKV (c) KIVI (d) ASVD (e) Palu (f) RVQ (g) CommVQ (h) CQ (i) **VQKV**

Figure 10: Detailed NIAH heatmaps of LLaMA3.1-8B. VQKV preserves exact retrieval under the main 82.8% compression setting.

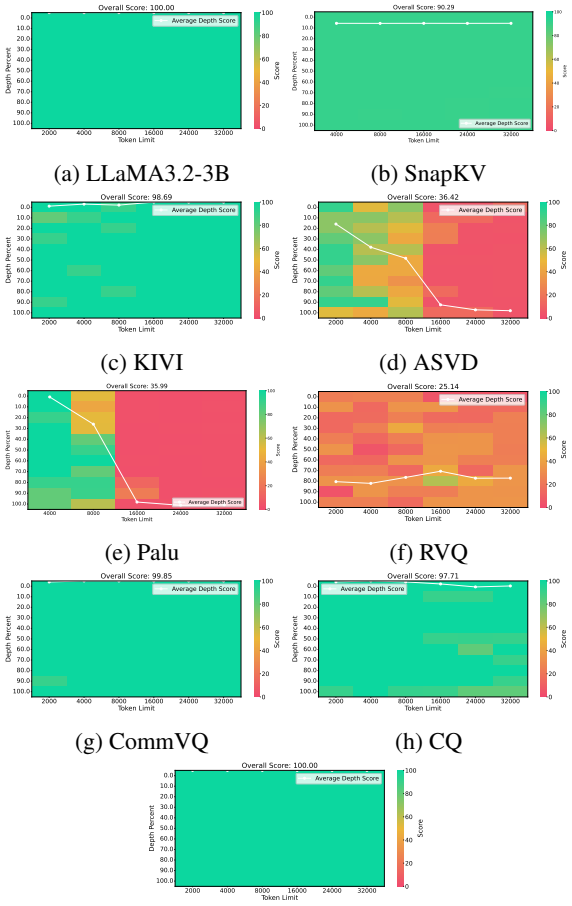
ment, consent, and ethics review board approval are not applicable.

A.7 Use of AI Assistants

The authors used AI assistants only for language polishing and grammar checking. The scientific claims, experimental design, implementation, data analysis, and final manuscript content were verified and are the responsibility of the authors. No AI assistant was used to generate experimental results or to make autonomous research decisions.

B Detailed NIAH Heatmaps

This appendix provides the detailed NIAH heatmaps referenced in the main long-context evaluation. The heatmaps (Figure 10, 11) complement the aggregate NIAH scores in Table 2 by showing retrieval accuracy across different context lengths and needle positions.



(a) LLaMA3.2-3B (b) SnapKV (c) KIVI (d) ASVD (e) Palu (f) RVQ (g) CommVQ (h) CQ (i) **VQKV**

Figure 11: Detailed NIAH heatmaps of LLaMA3.2-3B. VQKV preserves exact retrieval under the main 82.4% compression setting.

Ratio	Method	NIAH	RULER				
			4K	8K	16K	32K	**Avg.**
0	*LLaMA3.1-8B*	100.0	94.7	92.7	93.5	90.2	92.8
60.0%	KIVI	**100.0**	**94.8**	92.8	93.3	89.9	**92.7**
	ASVD	94.0	73.5	62.5	54.8	45.5	59.0
	Palu	**100.0**	90.5	85.3	82.8	78.4	84.3
	RVQ	63.9	40.2	28.4	26.4	20.2	28.8
	CommVQ	**100.0**	**94.8**	92.3	92.5	89.1	92.2
	CQ	**100.0**	94.7	**93.0**	92.5	**90.0**	92.6
	VQKV	**100.0**	94.6	92.9	**93.4**	**90.0**	**92.7**
80.0%	KIVI	**100.0**	92.3	89.8	**88.7**	**82.3**	**88.3**
	ASVD	3.3	0.9	0.9	0.2	0.2	0.6
	Palu	**100.0**	89.9	84.7	83.4	78.0	84.0
	RVQ	39.6	9.2	4.9	4.2	1.5	5.0
	CommVQ	94.9	69.7	62.6	55.5	46.4	58.6
	CQ	**100.0**	**93.6**	89.6	87.7	80.6	87.9
	VQKV	**100.0**	93.0	**90.0**	87.1	81.0	87.8
90.0%	KIVI	2.9	2.9	2.6	2.3	2.1	2.5
	ASVD	3.8	0.1	0.1	0.1	0.0	0.1
	Palu	88.8	66.1	56.0	51.1	42.1	53.8
	RVQ	3.6	0.3	0.4	0.6	1.2	0.6
	CommVQ	93.2	55.9	47.6	42.2	38.2	46.0
	CQ	46.0	33.2	29.0	21.6	16.5	25.1
	VQKV	**99.9**	**84.3**	**76.8**	**71.4**	**61.2**	**73.4**
95.0%	KIVI	2.7	2.0	1.9	1.8	1.5	1.8
	ASVD	2.5	0.1	0.0	0.0	0.1	0.0
	Palu	9.1	12.4	8.2	6.3	3.2	12.4
	RVQ	3.5	0.1	0.2	0.2	0.2	0.2
	CommVQ	23.4	14.7	10.5	11.5	10.0	11.7
	CQ	4.4	5.0	3.9	3.1	1.6	3.4
	VQKV	**37.8**	**43.2**	**22.7**	**16.4**	**13.0**	**23.8**

Table 3: Results of LLaMA3.1-8B on NIAH and RULER with different KV-cache reduction ratios. The **Ratio** column reports KV-cache reduction ratios.

C Detailed Results under Different KV-cache Reduction Ratios

Table 4 reports the detailed LongBench results of LLaMA3.1-8B under different KV-cache reduction ratios, and Table 3 reports the corresponding NIAH and RULER results. For each target ratio, we adjust the controllable compression hyperparameters of ASVD, Palu, RVQ, CommVQ, CQ, and VQKV to match the target ratio as closely as possible. For methods whose discrete settings do not exactly match a target ratio, we use the closest supported setting and report measured scores within the corresponding target-ratio block.

D Detailed Results in Ablation Study

We conduct ablation studies on the number and size of codebooks. On LLaMA3.1-8B (Grattafiori et al., 2024), we evaluate different combinations of codebook numbers and codebook sizes on LongBench (Bai et al., 2024), NIAH (Li et al., 2024a), and RULER (Hsieh et al., 2024). The detailed experimental results are presented in Table 5.

Ratio	Method	Single-Doc			Multi-Doc			Summary			Few-shot			Synthetic		Code		Avg.
		NQ	Qsp	MF	HQ	WQ	Msq	GR	QS	MN	TR	TQ	SS	PC	PR	LCC	Re-P	
0	*LLaMA3.1-8B*	13.0	20.4	32.1	12.0	14.0	8.7	29.7	25.2	0.9	73.5	91.0	47.2	0.8	26.8	72.0	69.2	33.5
60.0%	KIVI	12.4	20.0	**31.9**	**11.9**	14.2	8.8	**30.0**	**25.5**	0.9	73.5	**91.0**	**47.2**	0.9	27.2	71.9	69.5	**33.6**
	ASVD	3.0	17.1	17.4	9.8	11.2	6.8	18.6	20.0	11.5	63.0	86.1	41.5	1.8	16.1	40.4	46.1	25.7
	Palu	6.3	18.7	23.8	11.2	12.0	6.8	25.6	23.9	5.5	73.5	87.0	44.2	**2.1**	5.0	58.4	54.5	28.6
	RVQ	3.3	18.0	21.9	7.7	9.0	4.8	29.7	21.9	**13.5**	0.0	20.0	3.0	0.2	0.5	51.6	55.2	16.3
	CommVQ	11.3	**20.5**	30.5	11.7	**15.1**	**8.9**	29.2	24.4	0.9	73.5	**91.0**	46.4	1.1	**33.3**	71.3	68.4	**33.6**
	CQ	10.8	20.3	29.4	10.4	11.3	8.2	29.3	24.8	1.3	**74.5**	90.1	**47.2**	1.3	32.5	70.5	65.1	32.9
	VQKV	**12.8**	20.0	**31.9**	11.7	14.4	8.7	29.2	25.1	0.8	73.5	90.5	47.1	0.8	27.3	**72.5**	**69.7**	33.5
80.0%	KIVI	**14.1**	20.5	30.9	11.9	12.9	8.0	28.3	**24.9**	0.9	73.0	**91.3**	**46.4**	0.9	28.0	70.7	68.0	**33.2**
	ASVD	0.5	3.2	2.4	1.0	1.2	1.2	4.3	5.4	4.3	12.5	3.0	5.4	**3.1**	4.1	23.7	27.4	6.4
	Palu	6.7	18.2	22.7	11.7	11.9	6.5	24.6	23.6	5.6	73.5	87.8	44.8	2.4	5.1	56.6	54.0	28.5
	RVQ	1.7	8.7	13.2	4.6	6.8	3.1	20.0	13.5	**18.3**	2.0	25.1	14.5	0.1	0.5	31.7	38.2	12.6
	CommVQ	9.6	17.8	**34.7**	**17.4**	**14.8**	**10.0**	23.0	22.9	1.0	67.5	90.3	38.0	0.5	**31.5**	62.4	57.5	31.2
	CQ	9.4	**20.7**	22.2	10.0	11.5	7.8	**29.0**	24.3	0.8	**73.5**	90.1	45.7	1.6	6.6	65.2	61.8	30.0
	VQKV	**14.1**	19.7	29.9	11.7	13.7	8.2	26.5	23.7	0.9	**73.5**	90.5	45.9	0.8	25.7	**71.7**	**69.1**	32.8
90.0%	KIVI	2.8	5.7	5.3	6.7	4.3	3.5	6.7	5.5	0.4	32.8	14.2	5.5	1.4	11.5	30.6	27.5	10.3
	ASVD	0.6	2.4	2.1	0.5	0.6	0.7	4.6	8.8	**8.6**	0.0	2.3	8.8	0.6	2.5	19.8	20.6	5.2
	Palu	7.3	14.2	20.6	9.3	9.8	5.1	9.5	20.3	6.6	70.0	80.5	40.5	1.1	4.0	39.8	43.0	23.8
	RVQ	0.4	2.2	1.5	0.9	1.3	0.6	1.9	2.9	1.8	0.5	1.7	2.1	0.1	2.4	24.8	22.5	4.2
	CommVQ	10.6	15.5	**32.5**	**13.8**	**14.8**	**8.1**	18.2	22.9	0.6	64.5	89.8	41.1	0.5	**36.5**	62.3	57.4	30.6
	CQ	6.9	15.5	15.2	8.6	9.2	5.0	14.9	20.3	1.4	65.5	81.3	39.9	**1.2**	4.6	41.4	48.1	23.7
	VQKV	**13.5**	**17.8**	29.0	11.1	14.4	**8.1**	**20.6**	**23.6**	0.6	**72.5**	**90.8**	**45.6**	0.8	27.8	**71.5**	**68.1**	**32.2**
95.0%	KIVI	3.5	5.2	6.1	6.4	4.6	4.3	7.9	5.7	0.4	32.8	17.5	4.4	0.7	12.0	30.9	27.6	10.6
	ASVD	0.9	1.8	2.6	0.3	0.3	0.6	3.8	6.0	**7.2**	0.0	0.7	4.8	0.5	3.8	22.7	21.1	4.8
	Palu	3.3	6.8	10.2	4.2	3.8	2.2	5.2	14.6	6.1	37.0	13.3	16.7	2.2	3.6	22.9	25.0	11.1
	RVQ	0.3	1.3	1.0	0.4	0.5	0.3	1.7	5.1	2.2	0.0	0.5	4.5	0.0	1.8	14.3	9.8	2.7
	CommVQ	7.9	11.2	**26.5**	**11.5**	**15.4**	**7.3**	7.6	22.0	0.5	38.5	87.1	34.4	0.7	27.4	54.5	48.8	25.1
	CQ	1.6	5.9	7.0	4.5	3.8	2.4	8.2	14.0	**7.2**	13.0	25.7	9.5	**2.9**	3.6	29.5	34.0	10.8
	VQKV	**14.6**	**15.5**	25.5	10.2	12.6	7.2	**16.2**	**22.6**	0.6	**53.0**	**90.5**	**44.5**	0.5	**30.8**	**70.8**	**65.1**	**30.0**

Table 4: Results of LLaMA3.1-8B on LongBench with different KV-cache reduction ratios. The **Ratio** column reports KV-cache reduction ratios.

N^k	S^k	N^v	S^v	Ratio	LongBench	NIAH	RULER				
							4K	8K	16K	32K	Avg.
56	1024	16	128	83.6%	32.43	99.86	90.58	86.28	80.58	73.35	82.70
56	*1024*	*16*	*512*	*82.8%*	*33.10*	*100.00*	*92.70*	*89.50*	*86.90*	*79.90*	*87.30*
56	1024	16	2048	82.0%	33.40	100.00	93.96	89.55	87.02	79.42	87.49
56	1024	16	8192	81.3%	33.47	100.00	94.08	91.70	90.09	85.45	90.33
56	1024	16	32768	80.5%	33.47	100.00	94.27	91.56	90.31	85.39	90.38
56	1024	8	512	84.6%	31.39	95.94	74.43	65.60	56.85	47.78	61.16
56	*1024*	*16*	*512*	*82.8%*	*33.10*	*100.00*	*92.70*	*89.50*	*86.90*	*79.90*	*87.30*
56	1024	40	512	77.5%	33.64	100.00	94.45	92.03	92.01	86.96	91.36
56	64	16	512	88.3%	32.55	100.00	87.72	82.03	76.28	69.62	78.91
56	256	16	512	85.5%	32.97	100.00	92.60	88.47	84.50	76.91	85.62
56	*1024*	*16*	*512*	*82.8%*	*33.10*	*100.00*	*92.70*	*89.50*	*86.90*	*79.90*	*87.30*
56	4096	16	512	80.1%	33.07	100.00	93.30	89.93	87.54	81.17	87.99
56	8192	16	512	78.7%	33.11	100.00	93.29	90.19	87.94	81.32	88.18
8	1024	16	512	94.5%	30.07	37.51	42.94	22.56	16.61	12.71	23.71
24	1024	16	512	90.6%	32.33	99.86	84.48	76.49	71.34	60.75	73.27
40	1024	16	512	86.7%	32.85	100.00	92.53	87.75	83.91	74.55	84.68
56	*1024*	*16*	*512*	*82.8%*	*33.10*	*100.00*	*92.70*	*89.50*	*86.90*	*79.90*	*87.30*
72	1024	16	512	78.9%	33.19	100.00	93.43	90.18	87.48	81.22	88.08

Table 5: Detailed results of different combinations of codebooks of VQKV on LLaMA3.1-8B (Grattafiori et al., 2024). The **Ratio** column reports KV-cache reduction ratios.