PiCoKV: Dequantization-Free KV Cache Quantization via Binary Principal-Component Coding

Anonymous authors
Paper under double-blind review

Abstract

The growing key–value (KV) cache limits long-context large language model (LLM) inference through its memory footprint and bandwidth demand. Quantization reduces these costs, but attention typically still requires dequantizing or unpacking the stored KV states into wider numerical operands for computation, introducing conversion overhead and limiting the computational benefits of low-bit storage. To bridge this gap between compact storage and computation, we present **PiCoKV**, a training-free, data-aware KV-cache quantization method for dequantization-free attention on packed binary codes. PiCoKV factorizes keys and values into binary codes and compact decoding matrices. By folding these matrices into attention's query and output paths, PiCoKV performs both query–key scoring and value aggregation directly on packed codes through word-level bitwise operations, processing multiple binary entries in parallel without unpacking them into wider numerical operands. Evaluations on long-context benchmarks for Llama-3.1-8B-Instruct and Qwen3-8B show that PiCoKV achieves accuracy comparable to full attention and outperforms low-bit quantization baselines. The combination of compact storage and direct computation yields practical efficiency gains: PiCoKV achieves higher end-to-end decoding throughput than the baselines, reaching $3.45\times$ that of FlashAttention-2 at a 256K context while compressing the KV cache by up to $6.3\times$. Our code is available at https://anonymous.4open.science/r/PiCoKV-BFC1/.

1 Introduction

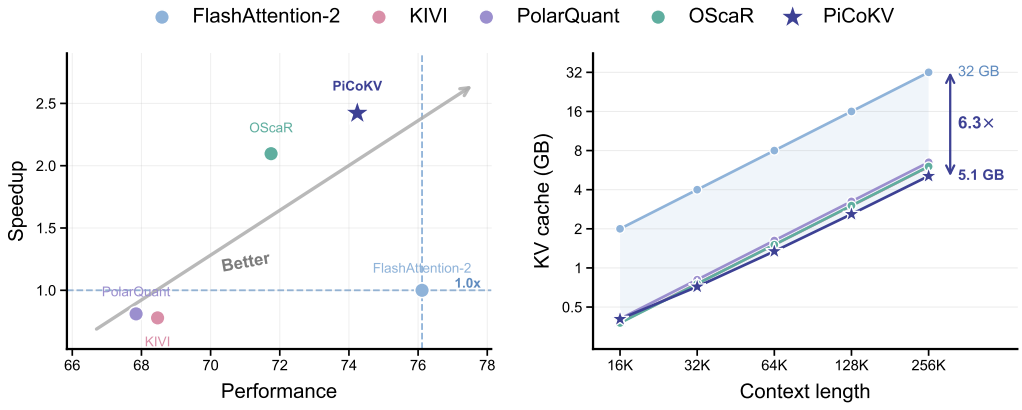

Figure 1: Accuracy–efficiency Pareto frontier on RULER at a 128K context (left) and KV-cache memory footprint from 16K to 256K (right).

Long-context large language models (LLMs) are increasingly important for document-level reasoning, retrieval, and agentic applications (Wang et al., 2024; Yang et al., 2025a; Grattafiori et al., 2024). Modern LLM inference consists of a prefill stage, where input tokens are processed in parallel, and a decoding stage, where output tokens are generated autoregressively. During decoding, each generated token attends to the fixed keys and values of all preceding tokens. To avoid recomputing these

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

activations at every step, the model stores them in a key–value cache (KV cache) that is initialized during prefill and extended during decoding. Because the KV cache grows linearly with sequence length, it becomes a dominant GPU-memory and bandwidth bottleneck for long-context inference.

KV-cache quantization reduces storage by retaining all tokens at lower precision (Liu et al., 2024; Hooper et al., 2024; Su et al., 2026; Kang et al., 2024). These methods, however, still require dequantizing or unpacking the cached data into numerical operands for attention, leaving a gap between compact storage and direct computation. A more computation-aware line reduces this conversion cost: PolarQuant turns query–key scoring into a table lookup (Wu et al., 2025), while QJL (Zandieh et al., 2025) and TurboQuant (Zandieh et al., 2026) provide data-oblivious inner-product estimators. These advances address query–key scoring, but none of these methods supports both query–key scoring and value aggregation directly on the packed KV representation.

To support both stages of attention directly on packed data, the KV representation must be compatible with the operations used in attention. Binary codes offer this compatibility through word-level bitwise operations for inner-product computation. BinaryPC (Yu et al., 2026) uses binary codes for efficient bitwise scoring directly on packed data to retrieve candidate tokens for sparse attention on the original KV states. This offers a useful starting point and raises a natural question: can binary codes serve as the KV cache itself, supporting both query–key scoring and value aggregation directly on packed data?

We answer this question affirmatively with **PiCoKV**, a training-free, data-aware KV-cache quantization method based on *dequantization-free code-domain attention*. It factorizes the keys and values into packed binary codes with compact decoding matrices; by folding the decoding matrices into attention's query and output paths, attention runs directly on the packed codes through word-level bitwise operations. These operations process multiple binary code entries in parallel without unpacking them into wider numerical operands. We evaluate whether this combination of compression and direct computation improves decoding efficiency while preserving model accuracy. Figure 1 previews PiCoKV's position on the accuracy–efficiency frontier for RULER (Hsieh et al., 2024): at a 128K context, it attains the largest decoding speedup among the compared methods while closely matching full attention, with the compression ratio growing to $6.3\times$ at 256K. Across multiple long-context benchmarks on Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen3-8B (Yang et al., 2025a), PiCoKV consistently outperforms the low-bit quantization baselines.

Our contributions are summarized as follows:

- **KV-Cache Quantization with Dequantization-Free Attention.** We introduce PiCoKV, a training-free, data-aware KV-cache quantization method that represents keys and values with packed binary codes and compact decoding matrices. By folding the decoding matrices into the query and output paths, PiCoKV supports both query–key scoring and value aggregation directly on packed codes without reconstructing full-precision KV states.
- **Efficient Bitwise Kernels and Prefill Pipeline.** We design dedicated CUDA kernels that perform code-domain attention through word-level bitwise operations, together with a prefill pipeline that encodes and bit-packs the keys and values. PiCoKV achieves $3.45\times$ the end-to-end decoding throughput of FlashAttention-2 (Dao, 2023) at a 256K context.

2 Related Work

2.1 KV-Cache Quantization

As context lengths grow, the KV cache becomes a dominant memory and bandwidth bottleneck, motivating KV-specific quantization methods: KIVI applies per-channel key and per-token value quantization with a full-precision residual (Liu et al., 2024), KVQuant moves key quantization before RoPE and adds non-uniform quantization and outlier handling (Hooper et al., 2024; Su et al., 2023), and QAQ formulates separate non-uniform strategies for keys and values (Cheng et al., 2025). Subsequent work refines bit allocation via importance- and sensitivity-aware mixed precision (He et al., 2024; Li et al., 2025), exploits cross-layer and coupled structure (Yang et al., 2025b; Zhang et al., 2024a), and reshapes the representation or reduces quantization error, as in OScaR and GEAR (Su et al., 2026; Kang et al., 2024). These methods improve fidelity and compression, but they still require dequantization during attention.

2.2 Attention over Quantized KV Caches

Reducing KV-cache storage does not automatically translate into proportional computational savings. Fusing dequantization into attention can avoid materializing intermediate KV states in global memory, but the kernel still unpacks and reconstructs numerical operands before multiplication (Du et al., 2026). The resulting arithmetic operates on these wider operands, so the compactness of the stored representation is not directly exploited for bit-level parallel computation.

An alternative is to avoid reconstruction by computing directly on the quantized states. QJL and TurboQuant compute query–key scoring through a data-oblivious inner-product estimator over the quantized keys, so full-precision keys are never materialized (Zandieh et al., 2025; 2026); PolarQuant instead reduces scoring to a table lookup over its finite quantized key states (Wu et al., 2025). These methods, however, avoid dequantization only in query–key scoring: value aggregation still operates on floating-point values, dequantized from the value cache. HACK instead computes both stages on quantized integer operands without reconstructing floating-point KV states, but it still requires unpacking the stored 2-bit KV data into INT8 before multiplication (Zhang et al., 2025).

In contrast, PiCoKV evaluates both query–key scoring and value aggregation on packed binary codes through word-level bitwise operations and population counts. These word-level operations process multiple binary code entries in parallel without unpacking them into wider numerical operands.

3 Method

To support both stages of attention directly on packed codes, PiCoKV uses a binary factorization that separates token-dependent codes from compact decoding matrices shared across tokens. Most tokens are stored in this binary representation, forming the global region of the KV cache. A small dense region retains sink and recent local tokens in uncompressed form for standard attention. For a single KV head, let $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ denote the keys and values in the global region, where N is the number of global tokens and d is the head dimension. PiCoKV approximates these states as

$$\mathbf{K} \approx \mathbf{C}_K \mathbf{D}_K, \qquad \mathbf{V} \approx \mathbf{C}_V \mathbf{D}_V, \tag{1}$$

where $\mathbf{C}_K, \mathbf{C}_V \in \{-1, +1\}^{N \times L_c}$ are binary codes of length L_c, and $\mathbf{D}_K, \mathbf{D}_V \in \mathbb{R}^{L_c \times d}$ are the decoding matrices. Figure 2 illustrates the overall pipeline.

3.1 Dequantization-Free Code-Domain Attention

Given the factorization in Eq. 1, PiCoKV folds the decoding matrices out of the context-sized contractions: the key decoding matrix moves to the query side, and the value decoding matrix is applied only after the token reduction, so full-precision global KV states (the $N \times d$ matrices) are never reconstructed; the only context-sized operands stored in the compressed cache are the packed binary codes $\mathbf{C}_K$ and $\mathbf{C}_V$.

Packed Query–Key Scoring. For a query row $\mathbf{q} \in \mathbb{R}^d$, this reorganization gives

$$\mathbf{q}\mathbf{K}^\top \approx \mathbf{q}(\mathbf{C}_K \mathbf{D}_K)^\top = \big(\mathbf{q}\mathbf{D}_K^\top\big)\mathbf{C}_K^\top. \tag{2}$$

In this packed scoring, the two operands are the projected query $\mathbf{q}\mathbf{D}_K^\top$ and the binary key-code matrix $\mathbf{C}_K$. The projection is computed once per query, so its cost is independent of context length. Let ℓ_t denote the resulting score for global token t, including the attention scale $1/\sqrt{d}$.

Packed Weight–Value Aggregation. The value path applies the same idea to the attention weights. Standard attention over the dense region provides a log-sum-exp (LSE) $\ell_{\mathcal{D}}$, which we use to shift the global logits before exponentiation for numerical stability, yielding the nonnegative, unnormalized weights $w_t = \exp(\ell_t - \ell_{\mathcal{D}})$. Let $\mathbf{n}$ denote the unnormalized weighted value sum, which is reordered as

$$\mathbf{n} = \mathbf{w}\mathbf{V} \approx \mathbf{w}(\mathbf{C}_V \mathbf{D}_V) = \big(\mathbf{w}\mathbf{C}_V\big)\mathbf{D}_V, \tag{3}$$

In this packed aggregation, the two operands are the attention weights $\mathbf{w}$ and the binary value-code matrix $\mathbf{C}_V$. Aggregation over tokens produces a code-domain accumulator of length L_c, to which $\mathbf{D}_V$ is applied once, so the decoding-matrix cost is independent of context length.

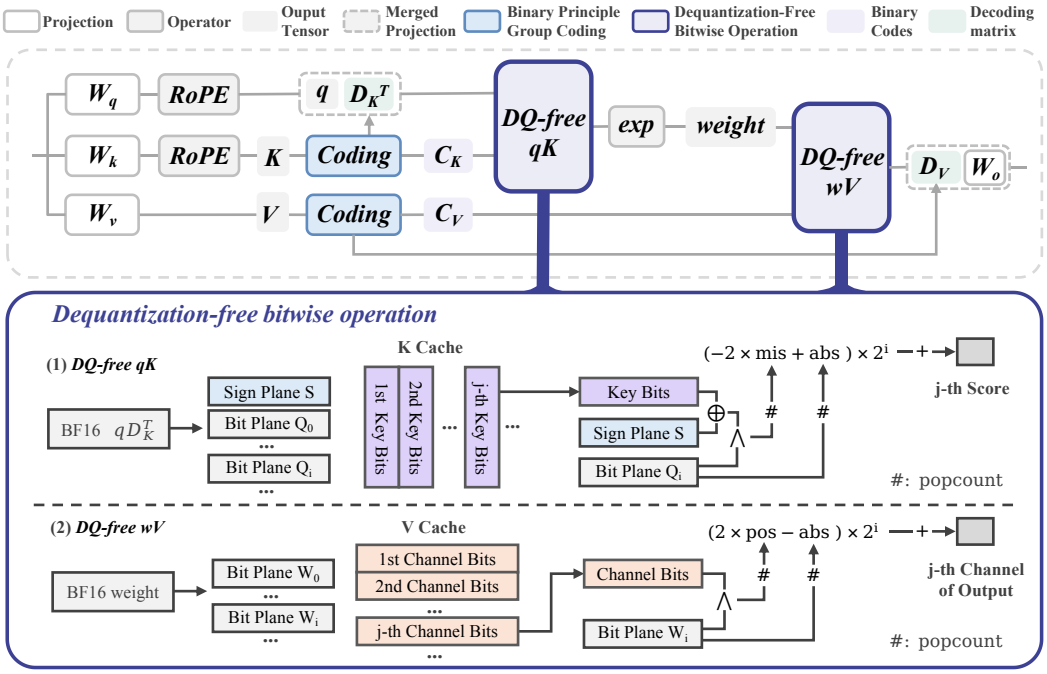

Figure 2: Conceptual overview of PiCoKV. Top: the overall pipeline—keys and values are coded into binary codes $\mathbf{C}_K$ and $\mathbf{C}_V$, and decoding performs dequantization-free (DQ-free) query–key scoring and weight–value aggregation. Bottom: the packed primitives—(1) query–key scoring (signed query) and (2) weight–value aggregation (unsigned weights), where *mis* and *abs* denote the mismatch and absolute-count terms computed by popcount, respectively.

Packed Bitwise Computation. Both contractions multiply a real-valued vector by a binary code matrix. PiCoKV quantizes the projected query and attention weights into signed and unsigned integers, respectively, and packs their bit planes to align with the cached codes—along the code dimension for scoring and the token dimension for aggregation. Both contractions then use word-level bitwise operations and popcount without unpacking the cached codes. Panels (1)–(2) of Figure 2 illustrate one representative bit plane for each contraction; Section 3.3 details the quantization, packing, and bitwise computation.

Unified Dense–Global Attention. The dense and global branches are combined under a single softmax rather than two separate normalizations. Standard attention over the dense region returns a normalized output $\mathbf{o}_{\mathcal{D}}$, while the global branch returns the unnormalized numerator $\mathbf{n}$ and mass $Z = \sum_t w_t$. Since the global weights are anchored to the dense log-sum-exp $\ell_{\mathcal{D}}$, the dense branch contributes unit mass under the same scaling, giving

$$\mathbf{o} = \frac{\mathbf{o}_{\mathcal{D}} + \mathbf{n}}{1 + Z}. \tag{4}$$

3.2 Binary Code Construction

The factorization in Eq. 1 assumes the binary codes and decoding matrices are already available. This subsection describes how to construct them efficiently for online compression while preserving fidelity for full-context attention. Exploiting the additive structure of the factorization, PiCoKV builds the representation progressively, fitting each block to the residual left by earlier blocks. Two considerations shape this construction: grouping binary components into one block to shorten the residual chain, and masking high-frequency RoPE components to reduce key-coding difficulty.

Grouped Progressive Coding. A single-component binary-coding procedure encodes one binary component at a time, fitting and removing its contribution from the residual before the next stage. Retrieval can tolerate a short code (Yu et al., 2026), but using the codes for both attention scoring and value aggregation calls for a larger code budget L_c, making the one-component-per-step sequential

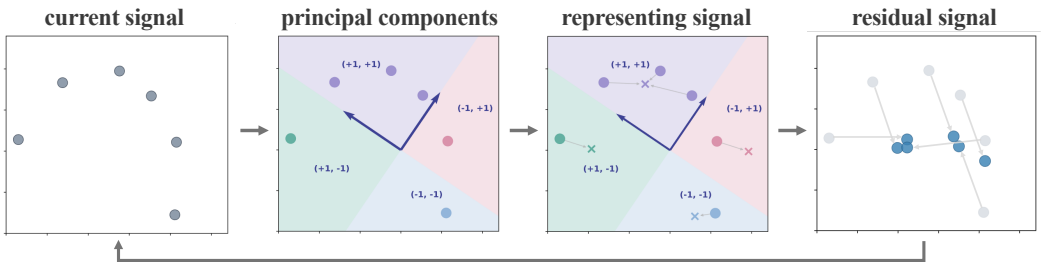

Figure 3: Illustration of one grouped residual coding stage. The three transitions correspond to binary-code construction, decoding-matrix fitting, and residual updating, respectively.

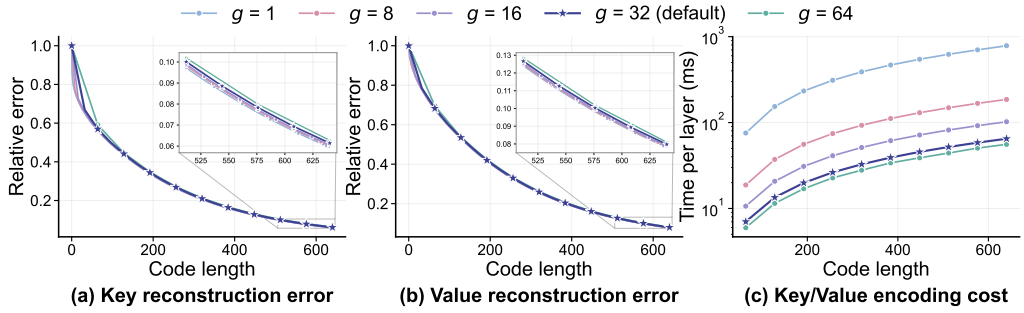

Figure 4: Reconstruction error and encoding cost as a function of coding-group size g and code length at a representative layer.

residual updates costly. PiCoKV instead fits g components jointly per block, shortening the residual chain from B to L_c/g stages. Figure 4 shows that larger groups substantially reduce encoding cost while leaving reconstruction error nearly unchanged.

For a fixed layer and KV head, let $\mathbf{X} \in \mathbb{R}^{N\times d}$ denote the global states to be coded, and let L_c and g denote the code length and block width, giving $G = L_c/g$ blocks. We initialize the residual $\mathbf{R}_0 = \mathbf{X}$, sampling a row subset $\mathcal{S}$ once for efficiency and tracking it across stages. At stage j, one step of randomized subspace iteration on the residual Gram matrix estimates the encoding matrix as

$$\mathbf{E}_j^\top = \mathrm{QR}\big(\widetilde{\mathbf{R}}_{j-1}^\top \widetilde{\mathbf{R}}_{j-1}\mathbf{Z}_j\big), \tag{5}$$

where $\widetilde{\mathbf{R}}_{j-1}$ denotes the residual restricted to the sampled rows $\mathcal{S}$, $\mathbf{Z}_j \in \mathbb{R}^{d\times g}$ contains g random orthogonal directions, and QR returns the orthonormal factor. The rows of $\mathbf{E}_j$ thus form an orthonormal basis that approximates the leading g-dimensional subspace of the current residual.

A sign projection of the residual onto $\mathbf{E}_j$ produces the binary block

$$\mathbf{C}_j = \mathrm{sign}\big(\mathbf{R}_{j-1}\mathbf{E}_j^\top\big) \in \{-1,+1\}^{N\times g}, \tag{6}$$

and the decoding matrix is fitted by ridge regression with penalty λ, minimizing the regularized reconstruction error of the current residual:

$$\mathbf{D}_j = \arg\min_{\mathbf{D}} \frac{1}{2N}\|\mathbf{R}_{j-1} - \mathbf{C}_j\mathbf{D}\|_F^2 + \frac{\lambda}{2}\|\mathbf{D}\|_F^2. \tag{7}$$

The reconstructed block $\mathbf{C}_j\mathbf{D}_j$ is then removed, updating the residual as $\mathbf{R}_j = \mathbf{R}_{j-1} - \mathbf{C}_j\mathbf{D}_j$. Concatenating the code blocks column-wise and the decoding matrices row-wise yields $\mathbf{C} \in \{-1,+1\}^{N\times L_c}$ and $\mathbf{D} \in \mathbb{R}^{L_c\times d}$, giving $\widehat{\mathbf{X}} = \mathbf{C}\mathbf{D}$. Algorithm 1 in Appendix A.4 summarizes this procedure, and Figure 3 illustrates one grouped coding stage.

RoPE-Aware High-Frequency Masking. For keys, code construction must also account for rotary position embedding (RoPE). KVQuant, for instance, quantizes keys before RoPE and rotates the dequantized keys during attention (Hooper et al., 2024). PiCoKV cannot take this route: its code-domain attention consumes the packed codes directly (Section 3.1), so a position-dependent rotation would force either materializing each key or computing a per-token query projection; the former loses the dequantization-free property, while the latter breaks the position-independent projection. PiCoKV therefore encodes the post-RoPE keys.

Encoding post-RoPE keys is harder. Figure 5 shows that the low-, middle-, and high-frequency groups exhibit similar singular-value decay before RoPE, but the high-frequency group decays much more slowly after RoPE, indicating that its energy is distributed across more singular directions. Representing this more dispersed energy requires greater coding capacity, making accurate key reconstruction more difficult under the fixed code budget in Eq. 1.

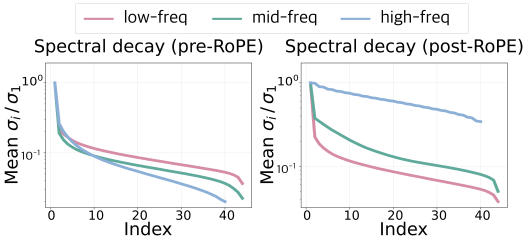

Figure 5: Singular-value decay of key channels across rotary frequency groups before RoPE (left) and after RoPE (right).

This cost is justified only if the highest-frequency components matter for long-range attention. Prior analyses find that the highest frequencies mainly encode short-range positional patterns and are under-utilized for long-distance attention (Barbero et al., 2025; Chiang & Yogatama, 2025). Since the dense branch retains the local tokens for those short-range patterns, we hypothesize that masking the highest-frequency components is nearly lossless for the global branch. Figure 6 supports this: applying HFM to full-precision (uncoded) global keys leaves accuracy essentially unchanged, while applying it to binary-coded global keys largely recovers the accuracy lost to key-coding errors.

Concretely, HFM zeros both coordinates of the r highest-frequency rotary pairs before binary coding, masking $2r$ channels in each global key. Dense keys and queries remain unchanged.

Online and Fixed Projection. PiCoKV supports online and fixed (offline) projection modes. In both modes the binary codes are computed from the current input during prefill; the modes differ only in the encoding and decoding matrices, which the online mode fits from the current input and PiCoKV-Fixed loads from offline calibration and reuses across inputs.

Cache Layout. After partitioning the historical cache into dense and global tokens, both modes encode the global keys and values using the binary coding procedure. Following BinaryPC (Yu et al., 2026), error-aware selection (EAS) keeps a small fraction of high-error global tokens uncompressed, where a token's error is the ℓ_2 norm of its final residual $\mathbf{R}_G$, computed separately for keys and values; this incurs a small memory overhead that grows linearly with context length. The remaining global codes are bit-packed, with the decoding matrices of each head stored. During decoding, the compressed global cache remains unchanged, while newly generated key/value states are appended to the dense cache.

3.3 Bitwise Operations for Attention Calculation

To realize the dequantization-free contractions of Section 3.1, PiCoKV evaluates both packed primitives with bitwise operations—XOR ($\oplus$), AND ($\wedge$), left shift ($\ll$), and popcount. We use 64-bit words, packing along the code dimension for scoring and the token dimension for aggregation.

Packed Query–Key Scoring. PiCoKV first quantizes the projected query $\mathbf{q}\mathbf{D}_K^\top$ into signed b_q-bit integers with scale $\Delta_q = \max_c |(\mathbf{q}\mathbf{D}_K^\top)_c|/(2^{b_q-1}-1)$, rounding each entry after division by Δ_q. An all-zero projected query is encoded as zeros. The sign is packed into sign packs S_p (bit 1 for negative entries and bit 0 otherwise), and the magnitude into $b_q - 1$ bit planes $Q_{i,p}$, where p indexes aligned packs of 64 code components and $i = 0, \ldots, b_q - 2$ indexes the magnitude plane.

With key codes packed as bit 1 for -1 and bit 0 for $+1$, the XOR mask $S_p \oplus \mathsf{k}_{t,p}$ marks sign disagreements, i.e., negative contributions, where $\mathsf{k}_{t,p}$ is the corresponding key-code pack for token t. ANDing this mask with each magnitude plane and counting the set bits yields the number of negative terms at that magnitude, so the token logit is

$$\ell_t = \frac{1}{\sqrt{d}} \Delta_q \sum_p \sum_{i=0}^{b_q-2} \Big(\mathrm{popcount}(Q_{i,p}) - 2\,\mathrm{popcount}\big((S_p \oplus \mathsf{k}_{t,p}) \wedge Q_{i,p}\big) \Big) 2^i, \tag{8}$$

Algorithm 2 in Appendix A.4 summarizes this computation.

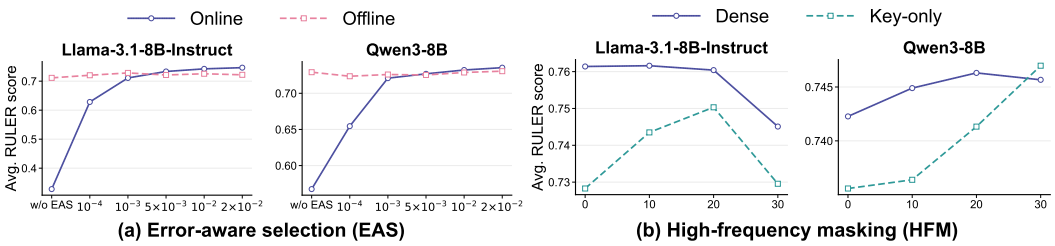

Figure 6: Ablation of EAS and HFM on RULER at 128K. (a) EAS varies the selection fraction for online and offline PiCoKV. (b) HFM sweeps masked high-frequency rotary pairs, comparing dense (uncoded) global keys with key-only coding, where only the keys are coded and values remain dense.

Packed Weight–Value Aggregation. PiCoKV quantizes the weights $w_t = \exp(\ell_t - \ell_{\mathcal{D}})$ into unsigned b_w-bit integers. Each pack p contains a set $\mathcal{T}_p$ of 64 tokens, with scale $\Delta_{w,p} = \max_{t\in\mathcal{T}_p} w_t/(2^{b_w}-1)$. Each weight is divided by this scale and rounded, and the resulting integers are packed into b_w weight bit planes $W_{i,p}$, where $i = 0, \ldots, b_w - 1$.

The value codes use the opposite convention, bit 1 for $+1$ and bit 0 for -1. For each code component c, the value pack $\mathsf{v}_{c,p}$ contains its codes for the same 64 tokens. ANDing this pack with a weight plane and counting ones gives the number of $+1$ codes at that weight magnitude, so the signed component contribution and the normalization mass are

$$n_c = \sum_p \Delta_{w,p} \sum_{i=0}^{b_w-1} \Big(2\,\mathrm{popcount}(\mathsf{v}_{c,p} \wedge W_{i,p}) - \mathrm{popcount}(W_{i,p})\Big) 2^i, \tag{9}$$

$$Z = \sum_p \Delta_{w,p} \sum_{i=0}^{b_w-1} \mathrm{popcount}(W_{i,p}) 2^i. \tag{10}$$

The numerator follows as $\mathbf{n} = \mathrm{concat}_c(n_c)\mathbf{D}_V$. Algorithm 3 in Appendix A.4 summarizes this computation.

4 Experiments

Table 1: RULER (Hsieh et al., 2024) average accuracy over 13 tasks for Llama-3.1-8B-Instruct and Qwen3-8B, evaluated at context lengths from 8K to 128K.

	Llama-3.1-8B-Instruct						Qwen3-8B					
Method	8K	16K	32K	64K	128K	Avg.	8K	16K	32K	64K	128K	Avg.
Full Attention	94.4	93.2	87.6	85.0	76.1	87.2	92.9	87.5	87.1	79.0	74.4	84.2
KIVI	93.2	92.4	87.5	82.5	68.5	84.8	89.3	83.3	80.1	72.0	65.5	78.0
PolarQuant	90.9	87.0	83.2	81.2	67.8	82.0	84.6	80.4	80.7	74.1	69.3	77.8
OScaR	93.9	93.0	87.6	84.0	71.7	<u>86.1</u>	90.6	85.2	82.9	73.9	69.5	80.4
PiCoKV	94.0	93.3	87.0	84.3	74.2	**86.6**	92.6	88.0	86.9	78.3	73.2	**83.8**
PiCoKV-Fixed	94.2	91.9	86.4	84.7	72.6	85.9	93.1	87.4	86.8	78.1	72.9	<u>83.7</u>

4.1 Setup

We evaluate PiCoKV on Llama-3.1-8B-Instruct (Grattafiori et al., 2024) and Qwen3-8B (Yang et al., 2025a), the former supporting 128K contexts natively and the latter 32K (extended to 128K via YaRN (Peng et al., 2024)); all experiments run on NVIDIA H800 GPUs. We compare against full attention (FlashAttention-2) and the low-bit quantization baselines KIVI (Liu et al., 2024), OScaR (Su et al., 2026), and PolarQuant (Wu et al., 2025), whose repository configurations are in Appendix A.1; result tables bold the best compressed method and underline the second best, with full attention as reference. We evaluate long-context understanding on LongBench, RULER, Needle-in-a-Haystack (Kamradt, 2023) and InfiniteBench, with benchmark descriptions in Appendix A.3.

PiCoKV uses Grouped Progressive Coding (GPC), with 320-bit codes for both keys and values (block width $g = 32$, ten blocks, i.e. about 2.5 bits per dimension versus 16-bit BF16). It retains

Table 2: Task-level results on InfiniteBench (Zhang et al., 2024b) for Llama-3.1-8B-Instruct and Qwen3-8B, averaged over 11 long-context tasks.

Method	En.Sum	En.QA	En.MC	En.Dia	Zh.QA	Code.D	Code.R	Math.F	R.Pass	R.Num	R.KV	Avg.
Llama-3.1-8B-Instruct												
Full Attention	28.59	31.28	68.00	19.00	36.99	0.00	4.00	50.00	100.0	98.00	46.00	43.81
KIVI	28.57	30.76	66.00	19.00	36.91	0.00	4.00	51.00	100.0	89.00	0.00	38.66
PolarQuant	26.98	28.91	66.00	17.00	36.79	0.00	5.00	51.00	100.0	79.00	1.00	37.43
OScaR	29.75	33.04	67.00	21.00	36.92	0.00	3.00	50.00	100.0	97.00	10.00	40.70
PiCoKV	28.76	31.57	68.00	19.00	36.17	0.00	3.00	50.00	100.0	98.00	28.00	**42.05**
PiCoKV-Fixed	28.12	32.13	68.00	18.00	36.60	0.00	4.00	50.00	100.0	99.00	18.00	41.26
Qwen3-8B												
Full Attention	28.73	17.29	69.00	14.00	28.07	9.00	0.00	53.00	100.0	100.0	9.00	38.92
KIVI	28.80	17.56	70.00	12.00	26.83	5.00	0.00	56.00	99.00	100.0	0.00	37.75
PolarQuant	26.80	16.87	70.00	12.00	27.94	6.00	0.00	56.00	100.0	100.0	1.00	37.87
OScaR	27.95	17.70	69.00	12.00	25.64	2.00	0.00	54.00	100.0	100.0	1.00	37.21
PiCoKV	28.64	16.79	70.00	13.00	28.77	11.00	0.00	54.00	100.0	100.0	8.00	**39.11**
PiCoKV-Fixed	28.33	16.38	70.00	12.00	27.98	10.00	0.00	53.00	100.0	100.0	6.00	38.52

64 sink tokens and 256 local tokens in dense form, sets the error-aware-selection (EAS) fraction to 0.01 (1% of global tokens), and applies high-frequency masking (HFM) to the 20 highest-frequency rotary pairs of each key head, with the remaining global tokens using packed binary codes. Unless otherwise noted, PiCoKV refers to the online variant, which fits the key and value encoding and decoding matrices (projections) per input, while PiCoKV-Fixed calibrates its projections once from 180 samples (Appendix A.2) and reuses them across inputs.

4.2 Main Results

Tables 1 and 2 report accuracy on RULER and InfiniteBench, respectively, while LongBench and Needle-in-a-Haystack results are reported in Appendix B.2 and Appendix B.3. Across these benchmarks, PiCoKV consistently tracks full attention and outperforms the strongest low-bit baseline, indicating that the dequantization-free, full-context code-domain attention largely preserves the model's long-context capability on the evaluated tasks.

Table 1 reports RULER results averaged over 13 tasks at context lengths from 8K to 128K, stress-testing retrieval and multi-hop reasoning as the context grows. PiCoKV attains 86.6 and 83.8 average accuracy on Llama and Qwen, staying within 0.6 points of full attention and outperforming the strongest low-bit baseline, OScaR, by up to 3.4 points. Its advantage widens with context length: at 128K, PiCoKV remains within 1.9 points of full attention, while the low-bit baselines degrade by up to 8.9 points. PiCoKV-Fixed stays within 0.7 points of the online variant on both models.

Table 2 reports task-level results on InfiniteBench, whose 11 tasks operate at contexts beyond 100K tokens. PiCoKV attains 42.05 and 39.11 on Llama and Qwen, slightly above full attention on Qwen (39.11 vs. 38.92) and staying within 1.76 points on Llama, while outperforming the strongest low-bit baseline by at least 1.24 points. PiCoKV-Fixed remains within 0.79 points of the online variant.

4.3 Efficiency

Figure 7 reports end-to-end decoding throughput under two scaling regimes: a context-length sweep with a batch size of one and a batch-size sweep at a 16K context. The context-length sweep extends beyond the models' 128K context to 256K on a synthetic KV cache, measuring raw end-to-end decoding throughput rather than benchmark accuracy; PiCoKV-Fixed's prefill cost closely matches FlashAttention-2 and is reported in Appendix B.1.

At a 16K context with batch size 16 (the batch-size sweep), PiCoKV raises end-to-end decoding throughput by $3.15\times$; at a 256K context with batch size one (the context-length sweep), the speedup reaches $3.45\times$. These gains follow from the same compression shown in Figure 1: the global KV cache is stored at about 2.5 bits per dimension (a $6.3\times$ smaller footprint at 256K), so each decoding step reads fewer bytes from the cache and skips dequantization entirely.

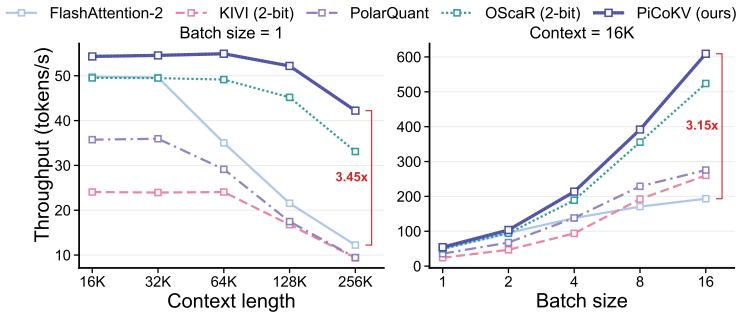

Figure 7: End-to-end decoding throughput of PiCoKV and the baselines on Llama-3.1-8B-Instruct. Left: context-length sweep with batch size one. Right: batch-size sweep at 16K context.

4.4 Ablation Studies

Figure 6 ablates HFM and EAS. HFM leaves accuracy with dense (uncoded) global keys essentially unchanged while allowing binary-coded global keys to largely recover the accuracy lost to key-coding error, suggesting that it reduces coding difficulty with little loss of useful attention information. An EAS fraction of 1% already provides most of the accuracy gains, with both projection variants showing little sensitivity to further increases.

Table 3: Ablation of PiCoKV on Llama-3.1-8B-Instruct on RULER (Hsieh et al., 2024) at 128K. Highlighted rows mark the default (320-bit codes, $g = 32$); group-width sweep varies g (K=V=320 bits) and code-length sweep varies K and V ($g = 32$). S-NIAH, M-Key, M-Q/V, and QA are task-group averages; VT, CWE, and FWE are individual tasks; Avg. averages all 13 tasks.

K bits	V bits	g (bits)	S-NIAH	M-Key	M-Q/V	VT	CWE	FWE	QA	Avg.
320	320	1	98.67	78.00	92.88	61.40	0.10	75.00	55.92	74.16
320	320	8	98.67	78.33	92.13	61.00	0.20	75.67	55.08	74.02
320	320	16	99.00	77.67	92.38	59.00	0.10	73.00	55.58	73.69
320	320	32	99.00	79.33	93.13	60.20	0.20	74.33	54.58	74.24
320	320	64	98.67	79.33	91.50	64.40	0.10	72.67	55.58	**74.26**
192	192	32	89.67	51.67	69.25	56.60	0.20	78.67	51.25	61.57
256	256	32	99.00	71.67	88.00	62.20	0.00	76.33	54.08	71.90
256	320	32	97.00	70.67	88.00	62.60	0.20	77.00	55.08	71.46
320	256	32	98.67	78.33	92.13	59.80	0.10	74.67	55.58	73.92
320	320	32	99.00	79.33	93.13	60.20	0.20	74.33	54.58	**74.24**

Table 3 ablates group width and code length around the default configuration (320-bit codes, $g = 32$) on Llama-3.1-8B-Instruct at 128K; Qwen3-8B results are in Appendix B.5. Group-width accuracy stays nearly flat from $g = 1$ to 64 (73.69–74.26), with $g = 32$ near-optimal on both models though per-model optima differ ($g = 64$ on Llama, $g = 8$ on Qwen). In the code-length sweep, accuracy improves with longer codes, the 320-bit setting reaching the best average (74.24); the gain is driven by the key code length (320/256 bits attains 73.92 vs. 71.46 for 256/320), consistent with post-RoPE keys being harder to code.

5 Conclusion

We present PiCoKV, a training-free, data-aware KV-cache quantization method that keeps global keys and values as packed binary codes and folds their decoding matrices into attention. This lets each decoding step score keys and aggregate values directly on packed codes through word-level bitwise operations, without unpacking them into wider numerical operands. High-Frequency Masking addresses hard-to-encode post-RoPE key components, while Grouped Progressive Coding shortens the residual-coding chain to reduce long-code construction cost. Across Llama-3.1-8B-Instruct and Qwen3-8B, PiCoKV closely tracks full attention on LongBench, RULER, and InfiniteBench while outperforming the evaluated low-bit quantization baselines. At a 256K context on an NVIDIA H800, it compresses the KV cache by up to $6.3\times$ and achieves $3.45\times$ the end-to-end decoding throughput of FlashAttention-2, showing that dequantization-free code-domain attention can translate KV compression into faster long-context decoding.

AI USE STATEMENT

Generative AI tools were used in this work for several supporting tasks, including translation, providing feedback on research methodology and experimental design, method implementation, code editing, literature summarization and analysis, and improving the clarity and readability of the manuscript. We did not use generative AI for qualitative or thematic data analysis, result interpretation, or the development of theoretical models or conceptual frameworks. Other disclosure categories, such as synthetic data generation, formulation of mathematical claims, development of key components for mathematical proofs, proof writing, hypothesis generation or refinement, and dataset cleaning or reformatting, were not applicable to this work.

All AI-assisted outputs were subject to human review and verification. Research suggestions and literature-related outputs were cross-checked against relevant published work through manual literature review, while AI-assisted code was manually examined and tested for correctness. Any AI-assisted manuscript content was further reviewed and revised by the authors for accuracy and consistency. The authors take full responsibility for the final manuscript and for all text, claims, code, and other artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We provide sufficient details to facilitate the reproduction of our results. The methodology and implementation of PiCoKV are described in the main text, including the KV-cache compression procedure and code-domain attention computation. Additional implementation details, experimental settings, hyperparameters, and ablation configurations are provided in the appendix. We also report the models, datasets, evaluation protocols, and baseline configurations used throughout our experiments. The source code and scripts required to reproduce the main experimental results will be made available in the supplementary materials.

REFERENCES

Zhangir Azerbayev, Edward Ayers, and Bartosz Piotrowski. Github repository: zhangir-azerbayev/proof-pile, 2022. URL `https://github.com/zhangir-azerbayev/proof-pile`.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. 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)*, pp. 3119–3137, 2024.

Federico Barbero, Alex Vitvitskyi, Christos Perivolaropoulos, Razvan Pascanu, and Petar Veličković. Round and round we go! what makes rotary positional encodings useful? In *International Conference on Learning Representations*, volume 2025, pp. 92408–92438, 2025.

Wen Cheng, Shichen Dong, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 2563–2571, October 2025.

Ting-Rui Chiang and Dani Yogatama. The rotary position embedding may cause dimension inefficiency in attention heads for long-distance retrieval. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 13552–13562. Association for Computational Linguistics, 2025.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning, 2023.

Dayou Du, Shijie Cao, Jianyi Cheng, Luo Mai, Ting Cao, and Mao Yang. Bitdecoding: Unlocking tensor cores for long-context llms with low-bit kv cache. In *2026 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–13. IEEE, 2026.

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.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification. In *Advances in Neural Information Processing Systems*, 2024.

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, pp. 1270–1303. Curran Associates, Inc., 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Gregory Kamradt. Needle in a haystack - pressure testing llms, 2023. URL `https://github.com/gkamradt/LLMTest_NeedleInAHaystack`.

Hao Kang et al. GEAR: An efficient KV cache compression recipe for near-lossless generative inference of LLM. *arXiv preprint arXiv:2403.05527*, 2024.

Xing Li, Zeyu Xing, Yiming Li, Linping Qu, Hui-Ling Zhen, Yiwu Yao, Wulong Liu, Sinno Jialin Pan, and Mingxuan Yuan. KVTuner: Sensitivity-aware layer-wise mixed-precision KV cache quantization for efficient and nearly lossless LLM inference. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

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 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 21–27 Jul 2024.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL `https://arxiv.org/abs/2104.09864`.

Zunhai Su, Rui Yang, Chao Zhang, Yaxiu Liu, Yifan Zhang, Wei Wu, Jing Xiong, Dayou Du, Xialie Zhuang, Yulei Qian, Yuchen Xie, Yik-Chung Wu, Hongxia Yang, and Ngai Wong. Oscar: The occam's razor for extreme kv cache quantization in llms and beyond. *arXiv preprint arXiv:2605.19660*, 2026.

Minzheng Wang, Longze Chen, Fu Cheng, Shengyi Liao, Xinghua Zhang, Bingli Wu, Haiyang Yu, Nan Xu, Lei Zhang, Run Luo, et al. Leave no document behind: Benchmarking long-context llms with extended multi-doc qa. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 5627–5646, 2024.

Songhao Wu, Ang Lv, xiao feng, Yufei zhang, Xun Zhang, Guojun Yin, Wei Lin, and Rui Yan. Polarquant: Leveraging polar transformation for key cache quantization and decoding acceleration. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 50584–50607. Curran Associates, Inc., 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Haoqi Yang, Yao Yao, Zuchao Li, Baoyuan Qi, Liu Guoming, and Hai Zhao. Xquant: Achieving ultra-low bit kv cache quantization with cross-layer compression. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 9796–9811, 2025b.

Daohai Yu, Zhanpeng Zeng, Keyu Chen, Wenhao Li, Zhifeng Shen, Luxi Lin, Ruizhi Qiao, Xing Sun, and Rongrong Ji. Training-free hashing-based attention via binary principal components. In *Forty-third International Conference on Machine Learning*, 2026.

Amir Zandieh, Majid Daliri, and Insu Han. QJL: 1-bit quantized JL transform for KV cache quantization with zero overhead. In *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, 2025.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *International Conference on Learning Representations*, 2026.

Peitian Zhang. Huggingface dataset: namespace-pt/long-llm-data, 2024. URL `https://huggingface.co/datasets/namespace-Pt/long-llm-data`.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. In *Advances in Neural Information Processing Systems*, volume 37, 2024a.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, et al. ∞bench: Extending long context evaluation beyond 100k tokens. *arXiv preprint arXiv:2402.13718*, 2024b.

Zeyu Zhang, Haiying Shen, Shay Vargaftik, Ran Ben Basat, Michael Mitzenmacher, and Minlan Yu. HACK: Homomorphic acceleration via compression of the key-value cache for disaggregated LLM inference. *arXiv preprint arXiv:2502.03589*, 2025.

A Implementation Details

A.1 Setup of Baseline Methods

We evaluate three training-free KV-cache compression baselines: KIVI, OScaR, and PolarQuant. The repository runners use the same Llama-3.1-8B and Qwen3-8B checkpoints as the main experiments.

KIVI. KIVI (Liu et al., 2024) uses asymmetric 2-bit quantization for both K and V. Keys use channel-wise grouping along the token dimension, and values use token-wise grouping along the head dimension. The runner uses group size 32 and keeps the most recent 128 K/V tokens in dense form.

OScaR. OScaR (Su et al., 2026) uses INT2 K-channel quantization with $g = 32$, Hadamard rotation, token-wise K-norm factoring, a 128-token packing block, and a 256-token residual eviction window. Its packed quantization-parameter tensors use FP32 storage, and the token-wise K norm uses one FP32 scalar per token. The Hadamard transform does not add persistent KV-cache entries.

PolarQuant. We use PolarQuant (Wu et al., 2025)'s mixed K/V path. Keys use 3-bit radius and 3-bit angle codes with group size 128; values use the KIVI-style groupwise value quantizer with 2-bit values and group size 128. The implementation keeps a 128-token K/V residual buffer. The radius/angle representation follows the repository's packed `uint8` layout. The V path reuses KIVI packing kernels.

A.2 Calibration Data

For offline calibration, PiCoKV-Fixed fits the key and value projections once from 180 samples spanning three corpora: PG19 (Rae et al., 2020) (literary text), ProofPile (Azerbayev et al., 2022) (mathematical reasoning), and CodeParrot (Zhang, 2024) (programming code), with 60 samples from each source.

A.3 Evaluation Benchmarks

LongBench. LongBench (Bai et al., 2024) evaluates general long-context understanding with heterogeneous natural-language and code tasks. We use the 16-dataset English/code subset in the repository runner: *narrativeqa*, *qasper*, and *multifieldqa_en* for single-document QA; *hotpotqa*, *2wikimqa*, and *musique* for multi-document QA; *gov_report*, *qmsum*, and *multi_news* for summarization; *trec*, *triviaqa*, and *samsum* for few-shot evaluation; *passage_count* and *passage_retrieval_en* for synthetic retrieval; and *lcc* and *repobench-p* for code completion. The runner applies the dataset-specific prompt templates and generation limits from its configuration files. We score QA with F1, summarization with ROUGE, classification with accuracy, synthetic retrieval with the corresponding retrieval/count metric, and code completion with code similarity, then average the 16 dataset scores uniformly.

RULER. RULER (Hsieh et al., 2024) tests controlled retrieval and reasoning behaviors at fixed context lengths. Our 13-task suite contains *niah_single_1*, *niah_single_2*, and *niah_single_3* for single-needle retrieval; *niah_multikey_1*, *niah_multikey_2*, and *niah_multikey_3* for multi-key retrieval; *niah_multiquery* and *niah_multivalue* for query/value retrieval; *ruler_vt*, *ruler_cwe*, and *ruler_fwe* for variable tracking and word extraction; and *ruler_qa_squad* and *ruler_qa_hotpot* for question answering. We run every task at 8K, 16K, 32K, 64K, and 128K context lengths, with a maximum generation budget of 128 tokens and the repository's fixed per-task evaluation limit. Main tables report the uniform average over all 13 tasks; this appendix provides the task-level tables.

InfiniteBench. InfiniteBench (Zhang et al., 2024b) evaluates long-context retrieval, reading, reasoning, and code behaviors with examples beyond 100K tokens. The runner uses 11 tasks: *passkey*, *kv_retrieval*, *number_string*, *code_run*, *code_debug*, *math_find*, *longdialogue_qa_en*, *longbook_qa_en*, *longbook_sum_en*, *longbook_choice_en*, and *longbook_qa_chn*. We set the model maximum length to 131,072 tokens, use zero-shot prompts, and follow the task-specific generation limits

in the benchmark templates. The runner uses the official task scorers, including exact or normalized matching for retrieval and multiple-choice tasks, token-level F1 for English QA, character level F1 for Chinese QA, numerical matching for math, and code-specific scoring for the programming tasks. We report the uniform average over the 11 task scores.

Needle-in-a-Haystack. Needle-in-a-Haystack (NIAH) (Kamradt, 2023) measures retrieval from long documents by placing a target fact at controlled relative depths. Our runner evaluates Llama-3.1-8B-Instruct and Qwen3-8B at 16 context lengths: 8K, 16K, 24K, 32K, 40K, 48K, 56K, 64K, 72K, 80K, 88K, 96K, 104K, 112K, 120K, and 128K tokens. For each length, it places the needle at ten depths from 0% to 100% in 11% increments and uses the same retrieval question and expected answer across methods. We report the fraction of depth and length settings for which the generated answer contains all required needle facts. The resulting depth-by-length accuracy matrices appear in Appendix B.

A.4 Algorithm Pseudocode

We present the pseudocode for the three core algorithms of PiCoKV. Algorithm 1 describes grouped progressive coding, which constructs the binary codes and decoding matrices from Section 3.2. Algorithms 2 and 3 specify the dequantization-free packed query–key scoring and weight–value aggregation computation from Section 3.3.

Algorithm 1 Grouped Progressive Coding

Input: Global states $\mathbf{X} \in \mathbb{R}^{N\times d}$; code length L_c; block width g; ridge penalty λ; row index set $\mathcal{S}$

$G \leftarrow L_c/g$; initialize residual $\mathbf{R}_0 \leftarrow \mathbf{X}$

$\widetilde{\mathbf{R}}_0 \leftarrow \mathbf{R}_0[\mathcal{S}, :]$

for $j = 1, \ldots, G$ **do**

  Draw g random orthogonal directions $\mathbf{Z}_j \in \mathbb{R}^{d\times g}$

  $\mathbf{E}_j^\top \leftarrow \mathrm{QR}\big(\widetilde{\mathbf{R}}_{j-1}^\top \widetilde{\mathbf{R}}_{j-1}\,\mathbf{Z}_j\big)$

  $\mathbf{C}_j \leftarrow \mathrm{sign}\big(\mathbf{R}_{j-1}\mathbf{E}_j^\top\big) \in \{-1,+1\}^{N\times g}$

  $\mathbf{D}_j \leftarrow \arg\min_{\mathbf{D}} \frac{1}{2N}\|\mathbf{R}_{j-1} - \mathbf{C}_j\mathbf{D}\|_F^2 + \frac{\lambda}{2}\|\mathbf{D}\|_F^2$

  $\mathbf{R}_j \leftarrow \mathbf{R}_{j-1} - \mathbf{C}_j\mathbf{D}_j$

  $\widetilde{\mathbf{R}}_j \leftarrow \widetilde{\mathbf{R}}_{j-1} - \mathbf{C}_j[\mathcal{S}, :]\mathbf{D}_j$

Concatenate blocks: $\mathbf{C} \leftarrow [\mathbf{C}_1, \ldots, \mathbf{C}_G]$, $\mathbf{D} \leftarrow [\mathbf{D}_1; \ldots; \mathbf{D}_G]$

Output: Binary codes $\mathbf{C} \in \{-1,+1\}^{N\times L_c}$ and decoding matrix $\mathbf{D} \in \mathbb{R}^{L_c\times d}$, giving $\widehat{\mathbf{X}} = \mathbf{C}\mathbf{D}$

Algorithm 2 Packed Query–Key Scoring

Input: Query $\mathbf{q} \in \mathbb{R}^d$; key-code packs $\mathsf{k}_{t,p}$ for token t and code pack p (bit 1 for -1, bit 0 for $+1$); decoding matrix $\mathbf{D}_K \in \mathbb{R}^{L_c\times d}$; attention scale $s_{\mathrm{attn}} = 1/\sqrt{d}$

Project and quantize $\mathbf{q}\mathbf{D}_K^\top$ to signed b_q-bit integers with scale Δ_q, and pack the result into sign packs S_p (bit 1 for negative, bit 0 for positive) and magnitude-bit packs $Q_{i,p}$, where $i = 0, \ldots, b_q - 2$

Precompute $\mathrm{abs}_{p,i} \leftarrow \mathrm{popcount}(Q_{i,p})$ for every pack p and magnitude plane i

for each global token t in parallel **do**

  for each aligned query–key pack pair p **do**

    $M_{t,p} \leftarrow S_p \oplus \mathsf{k}_{t,p}$

    $\mathrm{mismatch}_{t,p,i} \leftarrow \mathrm{popcount}(M_{t,p} \wedge Q_{i,p})$

    $s_{t,p} \leftarrow \sum_{i=0}^{b_q-2}\big((\mathrm{abs}_{p,i} - 2\,\mathrm{mismatch}_{t,p,i}) \ll i\big)$

  Add the pack scores: $s_t \leftarrow \sum_p s_{t,p}$

Concatenate the token scores: $\boldsymbol{\ell} \leftarrow \mathrm{concat}_t\big(s_{\mathrm{attn}}\Delta_q s_t\big)$

Output: Logits $\boldsymbol{\ell} \in \mathbb{R}^N$

Algorithm 3 Packed Weight–Value Aggregation

Input: Logits $\boldsymbol{\ell} \in \mathbb{R}^N$; dense LSE $\ell_{\mathcal{D}}$; value-code packs $\mathsf{v}_{c,p}$ for component c and token pack p (bit 1 for $+1$, bit 0 for -1); decoding matrix $\mathbf{D}_V \in \mathbb{R}^{L_c \times d}$

Compute $w_t = \exp(\ell_t - \ell_{\mathcal{D}})$ for every token t, where ℓ_t is the t-th entry of $\boldsymbol{\ell}$; within each token pack p, quantize the weights to unsigned b_w-bit integers with scale $\Delta_{w,p}$ and form weight-bit packs $W_{i,p}$, where $i = 0, \ldots, b_w - 1$

Precompute $\mathrm{abs}_{p,i} \leftarrow \mathrm{popcount}(W_{i,p})$ for every token pack p and weight plane i

for each code component c in parallel **do**

  for each aligned weight–value pack pair p in parallel **do**

    $\mathrm{positive}_{c,p,i} \leftarrow \mathrm{popcount}(\mathsf{v}_{c,p} \wedge W_{i,p})$

    $n_{c,p} \leftarrow \sum_{i=0}^{b_w-1} \big((2\,\mathrm{positive}_{c,p,i} - \mathrm{abs}_{p,i}) \ll i\big)$

  Add the pack contributions: $n_c \leftarrow \sum_p \Delta_{w,p} n_{c,p}$

Compute $Z \leftarrow \sum_p \Delta_{w,p} \sum_i (\mathrm{abs}_{p,i} \ll i)$

Concatenate the component contributions and apply the decoding matrix: $\mathbf{n} \leftarrow \mathrm{concat}_c(n_c)\mathbf{D}_V$

Output: Numerator $\mathbf{n} \in \mathbb{R}^d$ and mass $Z \in \mathbb{R}$

B Additional Experiment Results

B.1 Prefill Time

Figure 8 reports prefill time under the same two scaling regimes used for end-to-end decoding throughput: a context-length sweep with a batch size of one and a batch-size sweep at a 16K context. PiCoKV-Fixed's prefill cost closely tracks FlashAttention-2 and remains below the quantized baselines, which incur a visible prefill overhead at the largest measured points.

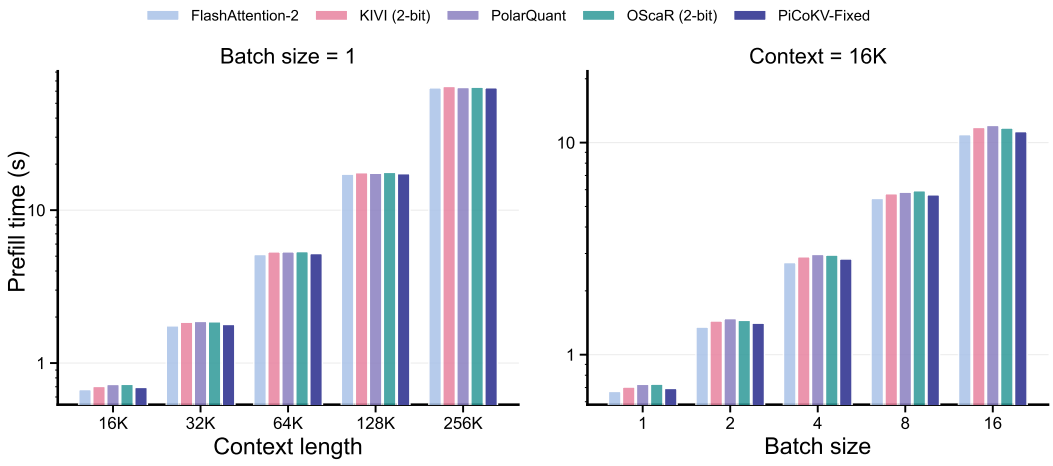

Figure 8: Prefill time of PiCoKV-Fixed and the baselines on Llama-3.1-8B-Instruct. Left: context-length sweep with batch size one. Right: batch-size sweep at 16K context.

B.2 LongBench Results

Table 4 reports LongBench results across 16 datasets spanning six task categories. PiCoKV stays within 0.06 points of full attention on both models and outperforms the strongest compressed baseline, OScaR, by up to 0.76 points. The offline PiCoKV-Fixed remains close, confirming that pre-calibrated projections retain nearly all of the online accuracy.

B.3 Needle-in-a-Haystack Retrieval

Figure 9 and Figure 10 visualize needle-in-a-haystack retrieval accuracy (Kamradt, 2023) as a function of document depth and context length for Llama-3.1-8B-Instruct and Qwen3-8B. PiCoKV pre-

Table 4: LongBench (Bai et al., 2024) results across 16 English and code datasets for Llama-3.1-8B-Instruct and Qwen3-8B.

	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Synthetic		Code		
Method	NQA	Qasper	MF	HPQA	2Wiki	Musique	Gov	QMSum	MNews	TREC	TQA	SAMSum	PC	PR	LCC	RB-P	**Avg.**
Llama-3.1-8B-Instruct																	
Full Attention	29.33	44.97	55.71	58.23	48.58	32.73	34.48	25.20	26.79	72.50	91.65	43.49	10.50	99.50	63.31	56.31	49.58
KIVI	30.07	44.65	54.74	57.97	48.28	32.51	34.52	25.66	26.59	72.50	91.76	43.46	9.40	99.50	61.93	55.11	49.29
PolarQuant	29.73	44.52	55.40	58.34	47.18	31.99	33.12	24.97	26.33	72.50	91.89	42.48	9.58	99.00	63.12	55.43	49.10
OScaR	30.24	43.72	56.31	57.25	48.46	32.59	34.33	25.17	26.66	72.50	92.32	43.92	10.57	99.50	62.33	55.90	49.49
PiCoKV	29.20	43.63	56.48	57.75	48.53	33.06	33.38	24.97	26.82	72.50	91.24	44.45	10.50	99.50	63.98	56.26	**49.52**
PiCoKV-Fixed	29.82	42.87	55.84	58.21	48.91	32.74	32.86	25.30	26.35	72.50	91.74	44.00	10.75	99.50	63.24	56.55	49.45
Qwen3-8B																	
Full Attention	24.93	47.96	51.20	57.82	44.11	29.94	33.64	23.54	24.49	71.50	90.73	44.24	5.89	97.50	68.42	61.38	48.58
KIVI	23.84	44.61	50.08	56.36	43.38	27.80	32.82	23.28	24.42	70.50	90.16	44.53	5.24	95.00	65.31	60.29	47.35
PolarQuant	24.19	44.17	48.79	56.53	42.58	28.56	31.88	24.22	24.99	70.00	89.40	44.31	6.08	93.50	60.60	56.34	46.63
OScaR	25.99	45.59	50.68	57.84	43.68	29.33	32.93	23.11	24.60	71.50	90.56	44.12	4.75	92.25	65.88	61.52	47.77
PiCoKV	25.69	46.43	51.58	57.81	44.61	29.60	33.28	23.57	24.32	71.50	90.73	43.83	7.00	97.00	68.25	61.27	**48.53**
PiCoKV-Fixed	24.87	46.55	51.00	58.69	44.32	29.35	33.18	23.41	24.52	71.50	90.98	43.95	5.75	98.00	68.07	61.46	48.48

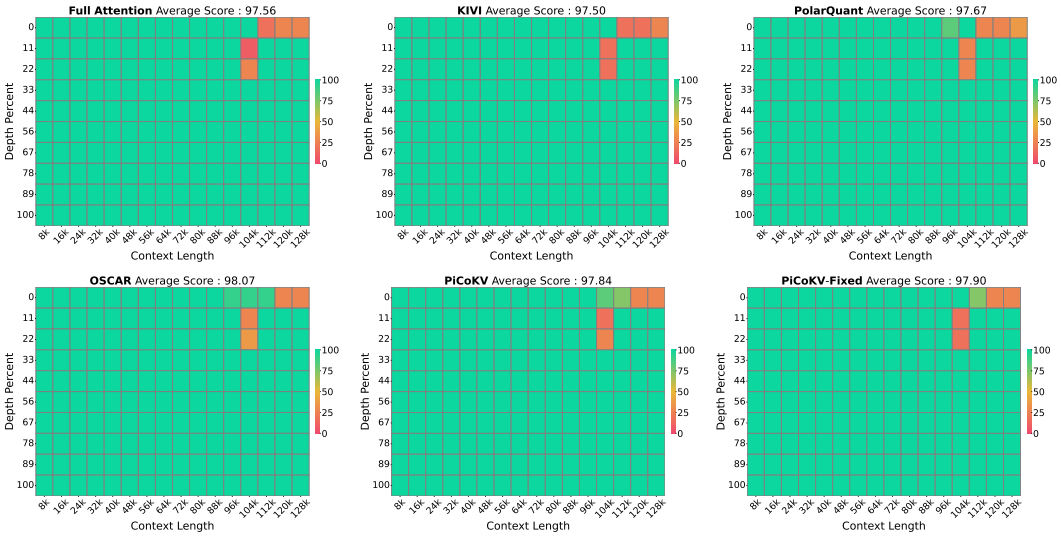

Figure 9: Needle-in-a-haystack retrieval accuracy (Kamradt, 2023) for Llama-3.1-8B-Instruct across document depths and context lengths. Panels show Full Attention, KIVI, PolarQuant, OScaR, PiCoKV, and PiCoKV-Fixed.

serves the full-attention retrieval pattern across depths and lengths, whereas the quantization baselines degrade more sharply on long contexts, especially on Qwen3-8B.

B.4 Task-level RULER Results

We report the full task-level RULER results in Tables 5–9 for context lengths from 8K to 128K. PiCoKV closely tracks full attention on retrieval (N-S1/2/3, N-MK1/2/3, N-MQ, and N-MV), variable tracking (VT), and question answering (QA-1/2) across all lengths, while the quantization baselines degrade more sharply as the context grows. The difference is clearest on the hardest multi-key retrieval task, N-MK3: at 128K on Llama-3.1-8B-Instruct, PiCoKV retains 56.0 accuracy versus 14.0 for KIVI, 10.0 for PolarQuant, and 25.0 for OScaR; on Qwen3-8B, PiCoKV retains 26.0 while the baselines drop to 0.0–5.0. The common-word extraction task CWE remains near zero for full attention on Llama-3.1-8B-Instruct at 128K and is low for every compressed method, indicating that its difficulty is largely a property of the base model rather than of KV-cache compression. PiCoKV-Fixed closely matches the online PiCoKV in average accuracy across all context lengths, with the

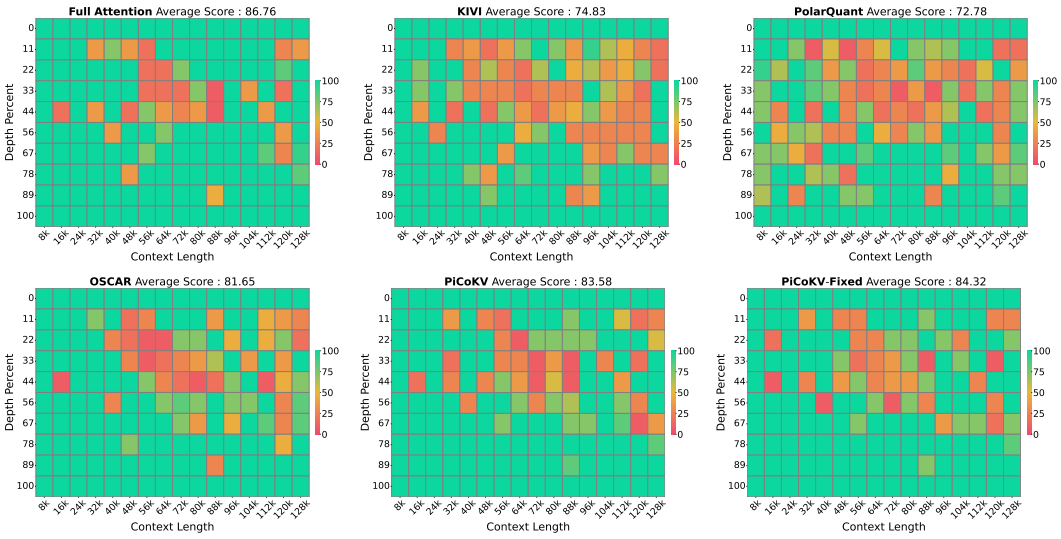

Figure 10: Needle-in-a-haystack retrieval accuracy (Kamradt, 2023) for Qwen3-8B across document depths and context lengths. Panels show Full Attention, KIVI, PolarQuant, OScaR, PiCoKV, and PiCoKV-Fixed.

Table 5: RULER (Hsieh et al., 2024) task-level accuracy at 8K context length for Llama-3.1-8B-Instruct and Qwen3-8B.

Method	N-MK1	N-MK2	N-MK3	N-MQ	N-MV	N-S1	N-S2	N-S3	CWE	FWE	QA-1	QA-2	VT	Avg.
Llama-3.1-8B-Instruct														
Full Attention	100.0	100.0	99.00	99.50	100.0	100.0	100.0	100.0	95.80	90.33	62.00	80.17	99.80	94.35
KIVI	100.0	99.00	95.00	99.50	99.50	100.0	100.0	100.0	93.30	87.67	61.00	80.17	97.00	93.24
PolarQuant	98.00	98.00	86.00	98.00	99.25	99.00	98.00	97.00	84.50	87.00	61.00	79.83	96.40	90.92
OScaR	100.0	99.00	99.00	99.50	100.0	100.0	100.0	100.0	93.70	88.67	62.00	79.50	99.40	93.91
PiCoKV	100.0	100.0	97.00	100.0	100.0	100.0	100.0	100.0	97.10	90.00	60.00	79.17	99.00	94.02
PiCoKV-Fixed	100.0	100.0	99.00	100.0	100.0	100.0	100.0	100.0	95.70	88.00	62.00	79.83	99.60	**94.16**
Qwen3-8B														
Full Attention	97.00	98.00	93.00	99.75	99.25	100.0	100.0	100.0	91.80	92.00	60.00	76.42	100.0	92.86
KIVI	96.00	96.00	70.00	99.25	99.25	100.0	99.00	100.0	89.40	86.00	59.00	74.42	92.80	89.32
PolarQuant	95.00	97.00	41.00	98.75	97.50	100.0	100.0	93.00	76.50	80.67	60.00	71.75	88.40	84.58
OScaR	97.00	97.00	83.00	99.25	99.75	100.0	100.0	100.0	88.60	90.33	57.00	76.42	89.20	90.58
PiCoKV	96.00	98.00	91.00	99.50	99.75	100.0	100.0	100.0	93.00	91.67	61.00	76.42	97.60	92.61
PiCoKV-Fixed	96.00	98.00	92.00	99.75	99.25	100.0	100.0	100.0	91.80	93.33	62.00	78.08	99.60	**93.06**

largest gap on the 128K multi-key retrieval task (N-MK3), confirming that offline calibration preserves overall task-level behavior.

B.5 Ablation on Qwen3-8B

Table 10 reports the ablation of PiCoKV on Qwen3-8B on RULER at 128K, mirroring the Llama-3.1-8B-Instruct ablation in Table 3. The findings are consistent: the default configuration attains the best code-length average, while accuracy remains nearly flat across the group-width sweep.

Table 6: RULER (Hsieh et al., 2024) task-level accuracy at 16K context length for Llama-3.1-8B-Instruct and Qwen3-8B.

Method	N-MK1	N-MK2	N-MK3	N-MQ	N-MV	N-S1	N-S2	N-S3	CWE	FWE	QA-1	QA-2	VT	Avg.
Llama-3.1-8B-Instruct														
Full Attention	100.0	100.0	98.00	97.75	100.0	100.0	100.0	100.0	86.90	95.33	57.00	78.17	98.60	93.21
KIVI	100.0	100.0	95.00	98.75	97.50	100.0	100.0	97.00	85.70	94.33	56.00	79.17	98.40	92.45
PolarQuant	98.00	97.00	92.00	97.75	97.50	98.00	98.00	100.0	30.60	94.00	56.00	76.83	95.60	87.02
OScaR	100.0	99.00	98.00	98.75	99.00	100.0	100.0	100.0	86.40	96.33	55.00	78.17	98.60	93.02
PiCoKV	100.0	100.0	98.00	99.50	100.0	100.0	100.0	100.0	85.70	95.67	57.00	78.83	98.00	**93.28**
PiCoKV-Fixed	100.0	100.0	98.00	98.25	100.0	100.0	100.0	100.0	71.60	94.67	57.00	77.50	98.20	91.94
Qwen3-8B														
Full Attention	96.00	84.00	74.00	99.75	95.50	100.0	100.0	100.0	84.50	94.33	46.00	66.42	97.40	87.53
KIVI	92.00	78.00	39.00	98.25	95.25	100.0	98.00	100.0	82.00	94.00	47.00	66.08	93.00	83.28
PolarQuant	91.00	85.00	18.00	99.25	96.50	100.0	100.0	97.00	69.30	84.67	45.00	68.08	91.40	80.40
OScaR	90.00	80.00	52.00	99.00	97.00	100.0	100.0	100.0	81.80	94.33	46.00	70.42	96.60	85.17
PiCoKV	96.00	85.00	76.00	99.50	95.25	100.0	100.0	100.0	86.10	94.00	46.00	67.75	98.40	**88.00**
PiCoKV-Fixed	95.00	85.00	74.00	99.25	95.50	100.0	100.0	100.0	84.40	94.33	46.00	64.75	98.00	87.40

Table 7: RULER (Hsieh et al., 2024) task-level accuracy at 32K context length for Llama-3.1-8B-Instruct and Qwen3-8B.

Method	N-MK1	N-MK2	N-MK3	N-MQ	N-MV	N-S1	N-S2	N-S3	CWE	FWE	QA-1	QA-2	VT	Avg.
Llama-3.1-8B-Instruct														
Full Attention	99.00	100.0	99.00	98.25	99.50	100.0	100.0	100.0	20.30	91.67	54.00	76.83	99.60	87.55
KIVI	99.00	97.00	87.00	97.25	96.25	100.0	100.0	100.0	42.20	93.00	54.00	75.83	95.60	87.47
PolarQuant	99.00	98.00	90.00	94.75	92.00	96.00	99.00	95.00	1.50	91.00	55.00	74.08	96.80	83.24
OScaR	99.00	100.0	97.00	96.75	95.50	100.0	100.0	100.0	28.40	92.00	56.00	75.83	98.60	**87.62**
PiCoKV	99.00	100.0	98.00	97.50	96.00	100.0	99.00	100.0	15.40	95.00	54.00	77.17	99.80	86.99
PiCoKV-Fixed	99.00	99.00	99.00	97.50	98.50	100.0	100.0	100.0	7.90	92.33	53.00	77.17	99.40	86.37
Qwen3-8B														
Full Attention	97.00	95.00	80.00	98.50	94.50	100.0	100.0	100.0	70.70	92.00	51.00	73.08	80.40	87.09
KIVI	93.00	77.00	34.00	97.75	94.75	99.00	97.00	97.00	68.50	91.00	51.00	64.75	76.00	80.06
PolarQuant	96.00	91.00	51.00	98.50	95.00	100.0	99.00	97.00	53.50	82.00	49.00	67.08	69.80	80.68
OScaR	93.00	93.00	47.00	98.75	94.50	100.0	99.00	99.00	68.20	91.33	52.00	70.75	70.60	82.86
PiCoKV	98.00	96.00	79.00	98.75	93.75	100.0	100.0	100.0	69.20	92.00	52.00	69.42	81.20	**86.87**
PiCoKV-Fixed	98.00	94.00	80.00	98.50	95.50	100.0	100.0	100.0	66.40	91.00	51.00	70.42	84.20	86.85

Table 8: RULER (Hsieh et al., 2024) task-level accuracy at 64K context length for Llama-3.1-8B-Instruct and Qwen3-8B.

Method	N-MK1	N-MK2	N-MK3	N-MQ	N-MV	N-S1	N-S2	N-S3	CWE	FWE	QA-1	QA-2	VT	Avg.
Llama-3.1-8B-Instruct														
Full Attention	99.00	99.00	97.00	99.50	98.75	100.0	100.0	100.0	1.30	86.67	51.00	75.08	97.20	84.96
KIVI	99.00	95.00	80.00	97.75	95.50	100.0	100.0	98.00	1.10	86.67	50.00	75.42	93.60	82.46
PolarQuant	98.00	94.00	77.00	97.75	91.00	99.00	99.00	100.0	0.10	86.33	49.00	72.08	92.00	81.17
OScaR	99.00	99.00	92.00	99.25	97.25	100.0	100.0	100.0	1.70	84.67	51.00	72.50	96.00	84.03
PiCoKV	98.00	98.00	97.00	99.50	97.75	100.0	100.0	100.0	0.20	87.00	49.00	72.75	97.20	84.34
PiCoKV-Fixed	99.00	98.00	97.00	99.25	98.00	100.0	100.0	100.0	0.70	85.33	52.00	74.08	97.60	**84.69**
Qwen3-8B														
Full Attention	90.00	65.00	33.00	99.00	93.75	100.0	100.0	100.0	54.60	82.00	47.00	65.75	97.20	79.02
KIVI	76.00	47.00	4.00	96.25	90.75	100.0	99.00	95.00	48.80	85.00	49.00	56.75	88.00	71.97
PolarQuant	88.00	54.00	9.00	96.25	94.50	100.0	97.00	98.00	45.70	81.00	46.00	61.42	91.80	74.05
OScaR	79.00	51.00	4.00	95.75	95.25	100.0	100.0	100.0	46.90	81.33	49.00	63.08	94.80	73.86
PiCoKV	89.00	69.00	30.00	97.75	93.50	100.0	100.0	100.0	45.10	83.00	47.00	66.42	97.20	**78.31**
PiCoKV-Fixed	91.00	63.00	32.00	98.25	95.00	100.0	100.0	100.0	47.20	81.33	47.00	64.75	95.40	78.07

Table 9: RULER (Hsieh et al., 2024) task-level accuracy at 128K context length for Llama-3.1-8B-Instruct and Qwen3-8B.

Method	N-MK1	N-MK2	N-MK3	N-MQ	N-MV	N-S1	N-S2	N-S3	CWE	FWE	QA-1	QA-2	VT	Avg.
Llama-3.1-8B-Instruct														
Full Attention	97.00	89.00	69.00	97.75	94.00	100.0	96.00	100.0	0.10	73.67	42.00	69.17	61.80	76.11
KIVI	97.00	67.00	14.00	94.75	87.75	100.0	95.00	96.00	0.00	71.00	40.00	68.17	59.40	68.47
PolarQuant	97.00	71.00	10.00	92.50	81.25	97.00	89.00	93.00	0.20	69.00	42.00	69.83	70.20	67.84
OScaR	97.00	84.00	25.00	96.25	91.25	99.00	96.00	99.00	0.20	72.67	40.00	70.17	62.20	71.75
PiCoKV	97.00	85.00	56.00	97.25	89.00	100.0	97.00	100.0	0.20	74.33	40.00	69.17	60.20	**74.24**
PiCoKV-Fixed	97.00	80.00	38.00	96.75	90.75	100.0	97.00	99.00	0.10	72.00	40.00	67.17	65.40	72.55
Qwen3-8B														
Full Attention	83.00	60.00	27.00	91.50	92.50	100.0	100.0	100.0	29.40	88.00	40.00	58.75	96.60	74.37
KIVI	73.00	39.00	0.00	85.25	80.75	100.0	97.00	90.00	21.80	86.33	37.00	49.75	92.20	65.54
PolarQuant	82.00	47.00	5.00	88.25	93.50	100.0	99.00	97.00	25.70	84.33	37.00	49.75	92.40	69.30
OScaR	80.00	46.00	4.00	90.00	91.25	100.0	97.00	97.00	24.20	87.33	37.00	56.75	93.60	69.55
PiCoKV	86.00	57.00	26.00	91.50	90.50	100.0	100.0	100.0	18.40	89.00	38.00	58.08	97.40	**73.22**
PiCoKV-Fixed	86.00	59.00	20.00	92.50	88.50	100.0	100.0	100.0	20.80	89.00	39.00	56.42	96.20	72.88

Table 10: Ablation of PiCoKV on Qwen3-8B on RULER (Hsieh et al., 2024) at 128K. Highlighted rows mark the default (320-bit codes, $g = 32$); group-width sweep varies g (K=V=320 bits) and code-length sweep varies K and V ($g = 32$). S-NIAH, M-Key, M-Q/V, and QA are task-group averages; VT, CWE, and FWE are individual tasks; Avg. averages all 13 tasks.

K bits	V bits	g (bits)	S-NIAH	M-Key	M-Q/V	VT	CWE	FWE	QA	Avg.
320	320	1	100.00	55.33	89.88	98.20	19.30	88.33	48.04	72.90
320	320	8	100.00	56.00	90.88	97.20	21.20	88.67	48.38	**73.35**
320	320	16	100.00	56.33	89.25	98.40	20.90	88.67	47.54	73.12
320	320	32	100.00	56.33	91.00	97.40	18.40	89.00	48.04	73.22
320	320	64	100.00	56.00	89.75	98.40	21.40	88.67	47.04	73.08
192	192	32	100.00	50.67	88.50	97.00	9.60	89.33	44.71	70.33
256	256	32	100.00	56.00	89.00	98.40	15.90	88.00	47.21	72.52
256	320	32	100.00	55.33	88.88	98.40	18.00	89.00	47.38	72.61
320	256	32	100.00	55.67	90.38	97.20	17.20	88.67	47.38	72.74
320	320	32	100.00	56.33	91.00	97.40	18.40	89.00	48.04	**73.22**