TAILORING THE QUANTIZATION SPACE FOR 1-BIT KV CACHE COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

The key-value (KV) cache becomes a major memory bottleneck in long-context LLM inference, placing substantial pressure on memory capacity and bandwidth. To mitigate this bottleneck, vector quantization (VQ) has emerged as a promising approach for aggressive KV cache compression. However, existing VQ methods degrade substantially in the 1-bit regime. At such extreme compression, each codebook must represent a larger group of channels with a limited set of centroids, making effective use of its capacity increasingly challenging. To address this, we introduce **TaSQ**, which tailors the VQ target space by combining query-guided channel weighting, cross-head normalization, and covariance-aware channel grouping to better reflect the error sensitivity and statistical structure of cached activations. Since these transforms are RoPE-compatible and can be easily merged into projection weights and codebooks, TaSQ preserves the conventional VQ lookup structure and adds negligible serving overhead. Across general, long-chain-of-thought reasoning, and long-context retrieval benchmarks, TaSQ consistently outperforms existing low-bit KV cache VQ baselines while preserving reasoning stability. On a single RTX 6000 Ada GPU, its SGLang implementation supports up to $14\times$ larger batch sizes and achieves $1.87\times$ higher peak throughput compared to the BF16 baseline.

1 INTRODUCTION

Modern LLM applications increasingly operate over interaction histories and large inputs, making long-context inference an important serving workload. Supporting such contexts requires storing the key and value activations of preceding tokens in the key–value (KV) cache. Because the cache grows linearly with sequence length and must be repeatedly accessed during autoregressive decoding, KV cache capacity and memory bandwidth become central bottlenecks in long-context serving.

KV cache compression directly targets these bottlenecks through various approaches, including token eviction (Zhang et al., 2023; Li et al., 2024), cache merging (Wang et al., 2024), and quantization (Liu et al., 2024; Hooper et al., 2024). Among them, quantization is one of the most widely studied and adopted approaches. In particular, vector quantization (VQ) is well suited to ultra-low-bit settings, as it jointly represents multiple channels with learned codewords that capture their multidimensional structure. Existing approaches learn codebooks over contiguous channel groups (Zhang et al., 2024) or normalize activations to use a calibration-free global codebook (Son et al., 2026). Yet preserving model quality as the rate approaches 1 bit per channel remains challenging.

To achieve high quantization quality at these rates, each codebook must effectively represent larger groups of channels with a limited set of centroids. We tackle this challenge by accounting for how quantization errors affect attention logits and how inter-channel dependencies can be exploited by VQ, based on two key insights. First, reconstruction errors in different channels have unequal effects on $QK^\top$. Second, VQ can exploit inter-channel dependencies only when the corresponding channels are represented by the same codebook.

Motivated by this, we introduce TaSQ (**Ta**ilored **S**pace Vector **Q**uantization), an LLM KV Cache VQ method that tailors the quantization space for accurate ultra-low-bit compression. TaSQ uses query-guided weighting to emphasize sensitive channels and covariance-aware grouping to assign dependent channels to the same local VQ group. A compact cross-head normalization further handles token-level magnitude outliers. The method employs only channel-wise scaling and permu-

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.

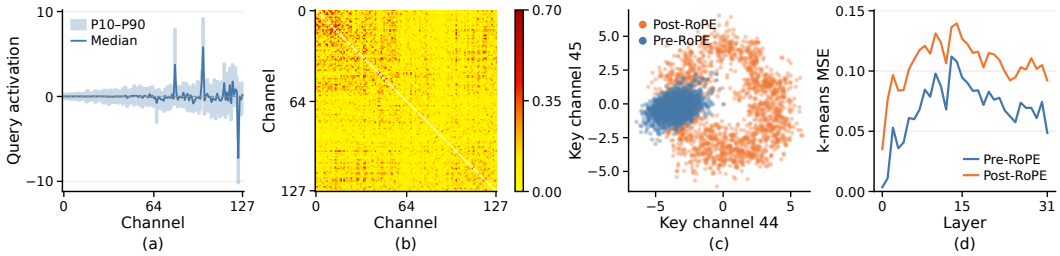


Figure 1: (a) Per-channel query medians and P10–P90 intervals. (b) Absolute inter-channel key correlations, with self-correlations masked. (c) Key distributions before and after RoPE for channels 44/45, using identical sampled tokens. (d) K-means reconstruction MSE on the fitting samples across all 32 layers, with matched group layouts and codebook sizes. All panels use a single 2,048-token GPQA window from Llama-3.1-8B; (a–c) show layer 15, KV head 0.

tation, without a dense rotation. These operations are mostly absorbed into the weight projection and codebooks, while runtime reconstruction operations are fused into the serving kernels. TaSQ therefore retains the conventional VQ lookup structure and incurs negligible runtime overhead.

Across general, reasoning, and long-context retrieval benchmarks, TaSQ substantially improves accuracy over existing VQ baselines in the 1-bit regime. In our SGLang serving implementation, TaSQ reduces the KV cache footprint and bandwidth demand, expanding the available KV cache pool by $12.48\times$ to support larger batches and enabling up to $1.87\times$ higher throughput compared to the full-precision BF16 baseline. These results demonstrate that tailoring the VQ target space is an effective approach to accurate and efficient ultra-low-bit KV cache compression.

2 Preliminaries

2.1 LLM Generation and KV Cache

Most modern large language models (LLMs) are based on the decoder-only Transformer architecture (Vaswani et al., 2017) and generate text autoregressively. Under causal masking, the keys and values of preceding tokens remain unchanged as new tokens are generated, allowing them to be stored in a key–value (KV) cache and reused across decoding steps. However, the cache grows linearly with sequence length and must be read at every decoding step. As context length increases, storing and accessing the KV cache places growing pressure on memory capacity and bandwidth.

2.2 Vector Quantization

Vector quantization (VQ) compresses a multi-dimensional vector by replacing it with the index of a codeword from a finite codebook. Given a codebook $\mathcal{C} = \{c_1, \ldots, c_K\}$ and an input $x \in \mathbb{R}^g$, VQ compresses the given input by assigning the nearest codeword:

$$z(x) = \arg\min_j \|x - c_j\|_2^2, \qquad \hat{x} = c_{z(x)}.$$

The codebook is typically learned by applying k-means to representative samples, with the resulting cluster centroids serving as codewords.

When a g-dimensional vector is represented by one of K codewords, the index cost is $\log_2 K/g$ bits per channel. Since VQ represents multiple channels jointly and can exploit structure in their joint distribution, it is particularly attractive for ultra-low-bit compression.

3 Motivation

Prior works have observed pronounced structure along the channel dimension of keys, including outlier channels and inter-channel dependencies (Liu et al., 2024; Hooper et al., 2024; Zhang et al., 2024; Xu et al., 2025). In this work, we likewise focus on the behavior of **key** channels. In particular,

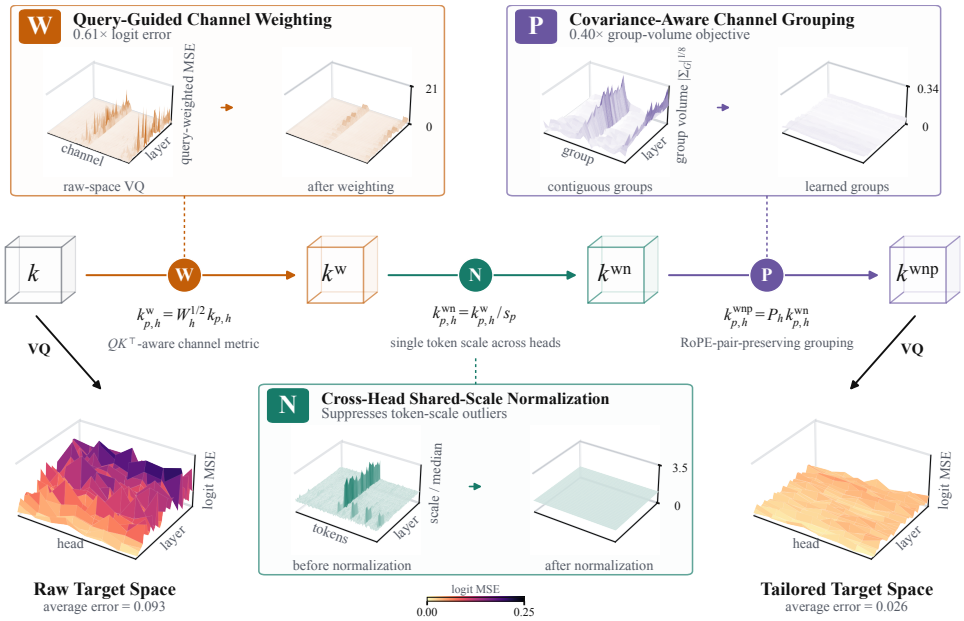


Figure 2: Overview of TaSQ. It tailors the VQ target space through query-guided channel weighting, cross-head shared-scale normalization, and covariance-aware grouping that preserves RoPE pairs, thereby reducing quantization error.

we observe that key channels differ in their sensitivity to queries, exhibit non-uniform correlations with one another, and undergo substantial distributional changes after RoPE. These observations motivate us to **tailor the quantization space to better reflect the structure of key channels**. In the matched analysis in Appendix C, values, in contrast, exhibit more uniform channel-wise loss sensitivity within each head and weaker average inter-channel correlations than keys. We therefore focus on keys and leave the design of a VQ space tailored to values to future work.

3.1 Sensitivity of Key Channels to Queries

Quantization errors in different key channels do not affect attention scores equally. In the query–key dot product, an error in each key channel is scaled by the corresponding query activation. As shown in Figure 1a, query activation distributions vary substantially across channels, with several channels exhibiting particularly large activation ranges. These differences imply that key channels have different sensitivities to quantization error, motivating the query-guided channel weighting introduced in Section 4.1.

3.2 Inter-channel Correlation

As shown in Figure 1b, key channels exhibit varying dependencies with one another. While some channel pairs are strongly correlated, others are nearly independent. Since VQ represents multiple channels jointly with a shared codebook, its ability to exploit these dependencies depends on which channels are grouped together. This motivates covariance-aware channel grouping, which we introduce in Section 4.3.

3.3 Pre-RoPE vs. Post-RoPE Keys

VQ is effective when the input distribution can be accurately represented by a finite set of centroids. However, RoPE rotates keys by position-dependent angles, substantially spreading their distribution. As shown in Figure 1c, the relatively compact pre-RoPE key distribution becomes widely dispersed after RoPE, making it more difficult for a shared codebook to cover the space efficiently. Figure 1d confirms this effect: pre-RoPE VQ achieves lower reconstruction error across all 32 layers, with a 35% lower total reconstruction error. Consequently, we design TaSQ to apply VQ to pre-RoPE keys.

4 METHOD

Overview. An overview of TaSQ is illustrated in Figure 2. During calibration, we apply query-guided channel weighting, normalization, and covariance-aware channel grouping to pre-RoPE keys, then fit a VQ codebook for each group in the resulting space. At inference time, keys are encoded as codebook indices and a scale, then reconstructed for attention with RoPE applied on the fly.

Notation. We use H for the number of KV heads and d for the head dimension. For KV head h, $k_{p,h}$ denotes the pre-RoPE key at position p, R_p its RoPE rotation matrix, and $\bar{q}_{t,h}$ a post-RoPE query at position t. Superscripts w, n, and p denote weighting, normalization, and permutation, respectively, and accumulate from left to right: k^{wnp} is the weighted, normalized, and permuted key. We denote the calibration corpus by $\mathcal{D}_{\mathrm{cal}}$ and use it as a superscript on expectations and covariances to indicate empirical estimation over this corpus. We omit the layer index since the same procedure is applied to all attention layers.

4.1 QUERY-GUIDED CHANNEL WEIGHTING

Motivated by the channel-wise variation in query distributions observed in Section 3.1, we derive channel weights from the query–key dot-product error and use them to transform keys into a weighted space.

Let $\delta_{p,h} = k_{p,h} - \hat{k}_{p,h}$ denote the pre-RoPE key quantization error at position p for KV head h. For a post-RoPE query $\bar{q}_{t,h}$ at position t, the squared dot-product error with the post-RoPE key is

$$\left(\bar{q}_{t,h}^{\top} R_p \delta_{p,h}\right)^2 = \delta_{p,h}^{\top} A_{t,p,h} \delta_{p,h}, \qquad A_{t,p,h} = R_p^{\top} \bar{q}_{t,h} \bar{q}_{t,h}^{\top} R_p.$$

To obtain a position- and token-independent summary of query–key sensitivity, we approximate $A_{t,p,h}$ by its average over valid causal query–key pairs in the calibration data:

$$\tilde{A}_h = \mathbb{E}_{t,p}^{\mathcal{D}_{\mathrm{cal}}}\left[A_{t,p,h}\right].$$

We then approximate $\tilde{A}_h$ with its diagonal to construct a channel-wise transform that preserves the native RoPE-pair structure:

$$W_h = \mathrm{diag}(\tilde{A}_h).$$

Appendix D provides further analysis of this approximation. This yields a surrogate for the expected squared dot-product error:

$$\mathbb{E}_{t,p}\left[\delta_{p,h}^{\top} A_{t,p,h} \delta_{p,h}\right] \approx \mathbb{E}_{t,p}\left[\delta_{p,h}^{\top} W_h \delta_{p,h}\right] = \mathbb{E}_{t,p}\left[\left\| W_h^{1/2} \delta_{p,h} \right\|_2^2\right]. \tag{1}$$

We therefore transform each pre-RoPE key into the weighted space:

$$k_{p,h}^{\mathrm{w}} = W_h^{1/2} k_{p,h}.$$

Since the expected squared reconstruction error in the weighted space is equal to the surrogate in Eq. 1, Euclidean VQ in this space incorporates query sensitivity directly into its reconstruction objective. Consistent with this interpretation, Figure 3(a) shows that query-guided weighting consistently lowers $QK^{\top}$ error and attention KL divergence across layers.

4.2 CROSS-HEAD SHARED-SCALE NORMALIZATION

Following Son et al. (2026), we apply token-wise normalization to suppress outlier tokens before VQ. While the prior work's head-wise normalization stores a separate scale for each token and KV head, we find that using a single RMS scale across all H KV heads is sufficient. For the weighted keys $k_{p,h}^{\mathrm{w}}$, we compute

$$s_p = \left(\frac{1}{Hd}\sum_{h=1}^{H} \|k_{p,h}^{\mathrm{w}}\|_2^2\right)^{1/2}, \qquad k_{p,h}^{\mathrm{wn}} = k_{p,h}^{\mathrm{w}} / s_p.$$

With b_s-bit scales, NSNQuant's head-wise scaling costs b_s/d bits per channel, whereas ours costs $b_s/(Hd)$, a factor-H reduction. We store s_p in FP16 ($b_s = 16$), retaining high scale resolution at a small rate overhead. Appendix D provides further analysis.

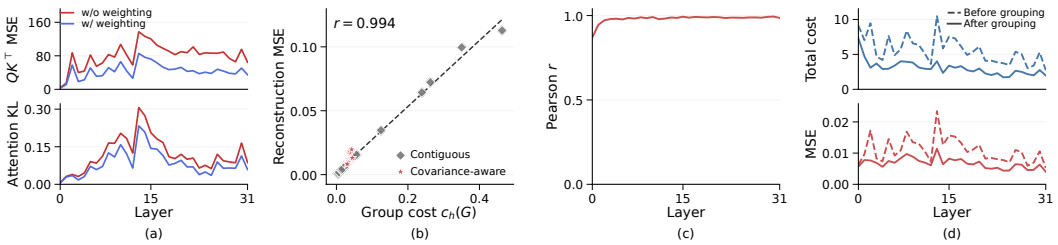


Figure 3: (a) $QK^\top$ MSE (top) and attention KL divergence (bottom) with and without query-guided weighting. (b) Relationship between the group cost $c_h(G)$ and VQ reconstruction MSE at layer 15, KV head 0. (c) Pearson correlation between group cost and reconstruction MSE across layers, averaged over 8 KV heads. (d) Total group cost (top) and VQ reconstruction MSE (bottom) under contiguous and covariance-aware grouping. Results are obtained with Llama-3.1-8B using disjoint GPQA samples for calibration and evaluation.

4.3 Covariance-Aware Channel Grouping

Motivated by the inter-channel dependencies observed in Section 3.2, we seek to group channels so that each VQ codebook can better exploit their joint structure. We therefore introduce a covariance-aware grouping criterion for the weighted and normalized key space.

For each KV head h, we estimate the covariance of the weighted and normalized keys from the calibration dataset:

$$\Sigma_h^{\mathrm{wn}} = \mathrm{Cov}^{\mathcal{D}_{\mathrm{cal}}}\left(k_h^{\mathrm{wn}}\right)$$

For a candidate channel group G, we define the following cost as a proxy for quantization error:

$$c_h(G) = \det\left(\Sigma_h^{\mathrm{wn}}[G,G] + \varepsilon I\right)^{1/|G|},$$

where $\Sigma_h^{\mathrm{wn}}[G,G]$ is the principal submatrix indexed by G, and $\varepsilon > 0$ is added for numerical stability.

This is motivated by the rate–distortion behavior of VQ. Under a Gaussian model, high-rate quantization theory gives the asymptotic scaling $D_G \propto K^{-2/|G|} \det(\Sigma_h^{\mathrm{wn}}[G,G])^{1/|G|}$ for the optimal mean squared reconstruction error of a K-centroid codebook (Gersho & Gray, 2012). Since $|G|$ and K are fixed across groups, the only group-dependent term is the covariance determinant, which motivates $c_h(G)$ as a proxy for quantization error.

Based on this cost function, we formulate a grouping problem as an optimization problem. Let g denote the number of channels quantized by each vector codebook. We seek a partition $\mathcal{G}_h$ of the d channels that minimizes

$$\min_{\mathcal{G}_h} \sum_{G \in \mathcal{G}_h} c_h(G), \qquad |G| = g,$$

subject to each RoPE pair belonging to the same group for efficient online RoPE application.

We approximately solve this grouping problem using **hierarchical matching**. Starting from individual RoPE pairs, we assign each pair of current groups a merge cost given by $c_h(G_a \cup G_b)$ and find a minimum-weight perfect matching. We merge the matched groups and repeat until each group contains g channels. Further details are provided in Appendix D.

The resulting partition defines a permutation matrix P_h that places the channels of each group in a contiguous block. We apply this permutation to the weighted and normalized keys:

$$k_{p,h}^{\mathrm{wnp}} = P_h k_{p,h}^{\mathrm{wn}} = P_h W_h^{1/2} k_{p,h} / s_p.$$

Empirical analysis. Figure 3(b–c) examines whether the covariance-aware cost serves as an effective proxy for actual VQ distortion. Despite depending only on group covariance, the proposed cost closely tracks the reconstruction MSE, with a Pearson correlation of $r = 0.994$ in the representative layer and consistently strong correlations across layers. Moreover, Figure 3(d) shows that optimizing this cost through covariance-aware grouping reduces both the grouping objective and the resulting VQ reconstruction error relative to contiguous grouping.

4.4 Vector Quantization in the Tailored Quantization Space

We apply VQ to the keys k^{wnp} lying in the weighted, normalized, and grouped quantization space. A separate K-centroid codebook $C_{h,G}$ is fitted for each layer, KV head, and channel group.

Fisher-weighted codebook fitting. While channel weighting captures *channel-wise* importance, calibration tokens can also differ in their influence on the language-modeling loss. To account for this *token-wise* variation, we adopt Fisher-weighted k-means following Zhang et al. (2024); Kim et al. (2023). Let $F_{p,h,i} = (\partial L/\partial k^{\mathrm{wnp}}_{p,h,i})^2$ denote the diagonal empirical Fisher, and let $\mathcal{G}$ denote the set of channel groups. We assign each group $G \in \mathcal{G}$ a Fisher weight by summing over its channels:

$$\omega_{p,h,G} = \sum_{i \in G} F_{p,h,i}.$$

We then fit the codebook $C_{h,G}$ using the calibration dataset with the weighted k-means objective:

$$C_{h,G} = \operatorname*{argmin}_{C} \sum_{p} \omega_{p,h,G} \left\| k^{\mathrm{wnp}}_{p,h,G} - c_{z_{h,G}(k^{\mathrm{wnp}}_{p,h,G})} \right\|_2^2.$$

At inference, each group is encoded by its nearest codeword, $z_{p,h,G} = \arg\min_j \|k^{\mathrm{wnp}}_{p,h,G} - c_{h,G,j}\|_2^2$. We store the resulting indices together with the shared scale s_p.

Runtime reconstruction. Let $W_{K,h}$ and $W_{Q,h}$ denote the key and associated query projection matrices, respectively. We absorb the weighting and permutation into the key projection, $\widetilde{W}_{K,h} = P_h W_h^{1/2} W_{K,h}$, and the inverse weighting into the decoder codebooks by transforming each codeword as

$$\widetilde{c}_{h,G,j} = \left[\left(P_h W_h P_h^\top \right)^{-1/2} \right]_{G,G} c_{h,G,j}.$$

We denote the resulting collection of transformed codebooks by $\widetilde{C}_h$. Given the stored VQ indices $z_{p,h}$ and shared token scale s_p, we reconstruct the key and compute the query–key dot product as

$$\hat{k}^p_{p,h} = s_p \widetilde{C}_h[z_{p,h}], \qquad \ell_{t,p,h} = \left(\widetilde{R}_{t,h} \widetilde{W}_{Q,h} x_t \right)^\top \widetilde{R}_{p,h} \hat{k}^p_{p,h},$$

where $\widetilde{W}_{Q,h} = P_h W_{Q,h}$ and $\widetilde{R}_{p,h} = P_h R_p P_h^\top$. Thus, both keys and queries remain in the permuted coordinate system throughout attention. Because P_h permutes complete RoPE pairs, $\widetilde{R}_{p,h}$ retains the standard block-diagonal RoPE structure; implementing it only requires reordering the RoPE frequencies according to the permuted pair layout.

The modified projection weights and decoder codebooks are materialized offline. During attention, codeword lookup, scale restoration, pairwise RoPE, and the query–key dot product are fused into the attention kernel. Consequently, TaSQ requires no runtime channel-shuffling operation, dense transformation, or additional kernel launch over standard VQ inference.

Value path. For values, we retain standard VQ with contiguous channel groups. As shown in Figure 6, values exhibit substantially less pronounced channel-wise characteristics than keys, making channel weighting and grouping less beneficial. Further analysis and value-side alternatives are provided in Appendix C.

5 Experiments

5.1 Experimental Setup

Models. We evaluate Llama-3.1-8B-Instruct (Grattafiori et al., 2024), Qwen3-4B, Qwen3-4B-Thinking-2507 (Yang et al., 2025), Deepseek-R1-Distill-Llama-8B (Guo et al., 2025) and Phi-4-reasoning-plus (Abdin et al., 2025). The first two models are used for general evaluation, and the latter three are used for reasoning. Long-context retrieval uses Llama-3.1-8B-Instruct, Qwen3-4B-Thinking-2507 and Phi-4-reasoning-plus.

Table 1: General performance under low-bit KV cache quantization. All results use greedy decoding. Bold denotes the best quantized result; higher is better.

Model	Method	Bits (K/V)	GSM8K	MATH500	MBPP	HumanEval	BBH	MMLU	Avg.
Llama-3.1-8B-Instruct	BF16	16.000/16.000	83.62	42.20	59.60	62.20	73.38	62.64	63.94
	CQ	1.250/1.250	68.01	19.60	52.00	54.27	40.08	54.21	48.03
	NovaKV	1.375/1.250	76.95	29.40	55.00	53.66	48.46	56.43	53.32
	NSNQuant	1.238/1.238	73.39	31.40	50.20	57.93	48.57	**58.33**	53.30
	TaSQ	1.266/1.250	**81.73**	**34.00**	**58.40**	**58.54**	**64.28**	**58.33**	**59.21**
Qwen3-4B	BF16	16.000/16.000	86.28	72.60	64.00	81.71	78.03	74.72	76.22
	CQ	1.250/1.250	70.36	60.80	45.40	65.85	52.08	64.14	59.77
	NovaKV	1.375/1.250	84.53	66.80	**62.60**	76.22	**68.80**	67.21	71.03
	NSNQuant	1.238/1.238	65.58	52.40	46.20	67.68	49.57	63.29	57.45
	TaSQ	1.266/1.250	**85.44**	**69.80**	**62.60**	**77.44**	68.28	**71.26**	**72.47**

Table 2: Long-CoT reasoning performance under low-bit KV cache quantization. We report the mean and standard deviation over three seeds. Bold denotes the best quantized result; higher is better.

Model	Method	Bits (K/V)	AIME'24	AIME'25	LCB-v6	SciBench	Avg.
Qwen3-4B-Thinking-2507	BF16	16.000/16.000	74.44±1.92	74.44±5.09	45.85±0.62	73.99±0.43	67.18±1.37
	CQ	1.250/1.250	26.67±3.33	14.44±5.09	18.99±0.44	54.00±0.94	28.52±1.54
	NovaKV	1.375/1.250	7.78±1.92	6.67±3.33	9.16±0.38	27.94±2.08	12.89±1.10
	NSNQuant	1.238/1.238	44.44±6.94	38.89±1.92	34.66±0.33	68.69±0.55	46.67±1.81
	TaSQ	1.266/1.250	**68.89±1.92**	**56.67±3.33**	**43.00±0.47**	**73.65±0.60**	**60.55±0.98**
DeepSeek-R1-Distill-Llama-8B	BF16	16.000/16.000	53.33±3.33	31.11±1.92	39.68±0.56	38.49±0.30	40.65±0.98
	CQ	1.250/1.250	26.67±3.33	26.67±5.77	23.29±0.45	35.45±1.61	28.02±1.72
	NovaKV	1.375/1.250	35.56±5.09	18.89±3.85	20.98±0.65	33.62±0.44	27.26±1.61
	NSNQuant	1.238/1.238	44.44±5.09	24.44±5.09	32.67±1.15	34.83±1.64	34.10±1.87
	TaSQ	1.266/1.250	**48.89±6.94**	**31.11±1.92**	**32.95±0.61**	**39.11±1.50**	**38.02±1.85**
Phi4-14B-Reasoning-Plus	BF16	16.000/16.000	71.11±1.92	66.67±3.33	46.60±0.71	48.22±1.03	58.15±1.01
	CQ	1.250/1.250	52.22±3.85	31.11±5.09	13.33±0.55	33.14±1.34	32.45±1.64
	NovaKV	1.375/1.250	41.11±10.18	35.56±1.92	21.93±0.29	41.57±1.86	35.04±2.63
	NSNQuant	1.238/1.238	63.33±6.67	55.56±5.09	37.12±1.04	44.99±0.65	50.25±2.12
	TaSQ	1.263/1.250	**71.11±5.09**	**60.00±0.00**	**41.20±0.52**	**50.58±1.90**	**55.72±1.36**

Benchmarks. We evaluate general performance on GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2024), MBPP (Austin et al., 2021), HumanEval (Chen et al., 2021), BBH (Suzgun et al., 2023) and MMLU (Hendrycks et al., 2020) with greedy decoding. Reasoning performance is evaluated on AIME 2024, AIME 2025, LiveCodeBench v6 (Jain et al., 2024), and SciBench (Wang et al., 2023) with thinking enabled and three sampling seeds; we report mean and standard deviation. Long-context retrieval is evaluated with RULER (Hsieh et al., 2024) needle-in-a-haystack at 4k, 8k, 16k, 32k, and 64k using 200 examples and three seeds per setting.

Calibration. CQ (Zhang et al., 2024), NovaKV (Fernández-Menduiña et al., 2026), and TaSQ use the same 64 calibration windows of 2,048 tokens: 48 from `gpqa_main` (Rein et al., 2023) and 16 from `codeparrot/codeparrot-clean` (CodeParrot, 2022). The same windows are used for activation collection and, for CQ and TaSQ, Fisher estimation. NSNQuant (Son et al., 2026) uses its calibration-free global codebook.

Decoding and residual policy. General tasks use greedy decoding and retain the most recent 64 tokens in full precision. Reasoning tasks use temperature 0.6, top-p 0.95, and generation limits up to 32,768 tokens; the first 64 and most recent 256 tokens remain in full precision. For reasoning and retrieval tasks, we report the mean and standard deviation over three sampling seeds. Each benchmark uses the same policy for all quantized methods.

Baselines and bit accounting. We compare TaSQ with the pre-RoPE VQ baseline CQ and the post-RoPE VQ baselines NSNQuant and NovaKV. We use the published 1-bit-regime configurations of CQ and NSNQuant. Because NovaKV does not report a corresponding configuration, we retain its original quantizer design and match its K codebook to the 1,024 centroids used by CQ and TaSQ. The resulting K/V rates are reported in each result table. Appendix B provides the full accounting.

Implementation and hardware. We implement all methods in SGLang (Zheng et al., 2024) with Triton attention kernels. All efficiency experiments are run on NVIDIA RTX 6000 Ada GPUs.

Table 3: Needle-in-a-haystack retrieval performance at increasing context lengths. We report the mean and standard deviation over three seeds. Bold denotes the best quantized result; higher is better.

Model	Method	Bits (K/V)	4k	8k	16k	32k	64k
Llama-3.1-8B-Instruct	BF16	16.000/16.000	98.50±1.73	98.83±0.76	98.67±0.29	98.67±0.76	98.17±1.04
	CQ	1.250/1.250	19.33±1.61	15.50±3.61	12.67±1.26	12.83±3.18	6.33±1.26
	NovaKV	1.375/1.250	84.83±0.29	81.67±3.18	64.00±2.18	21.33±2.25	1.83±0.29
	NSNQuant	1.238/1.238	89.17±1.61	86.00±1.32	86.83±4.25	88.17±1.15	82.17±0.58
	TaSQ	1.266/1.250	**93.33±2.08**	**93.17±0.76**	**92.00±1.32**	**93.67±1.04**	**93.17±0.76**
Qwen3-4B-Thinking-2507	BF16	16.000/16.000	100.00±0.00	100.00±0.00	99.83±0.29	99.67±0.29	95.33±2.36
	CQ	1.250/1.250	71.50±1.73	63.33±4.01	48.17±0.29	21.83±1.04	10.50±2.78
	NovaKV	1.375/1.250	**99.83±0.29**	27.67±2.25	0.00±0.00	0.00±0.00	0.00±0.00
	NSNQuant	1.238/1.238	97.83±2.47	97.00±0.50	93.67±0.76	91.17±0.29	73.00±2.18
	TaSQ	1.266/1.250	**99.83±0.29**	**98.67±0.76**	**98.83±0.58**	**98.50±0.00**	**81.50±1.50**
Phi4-14B-Reasoning-Plus	BF16	16.000/16.000	99.83±0.29	99.83±0.29	99.67±0.29	99.50±0.00	–
	CQ	1.250/1.250	62.00±1.32	54.83±3.33	47.67±3.51	34.33±4.25	–
	NovaKV	1.375/1.250	97.33±1.26	91.83±1.61	62.50±3.28	0.00±0.00	–
	NSNQuant	1.238/1.238	98.67±0.76	98.17±1.44	98.00±0.50	92.67±1.26	–
	TaSQ	1.263/1.250	**99.83±0.29**	**99.33±0.76**	**98.67±0.58**	**97.33±0.76**	–

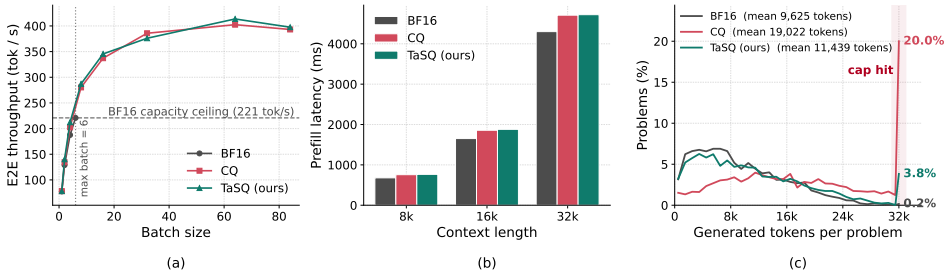


Figure 4: Serving efficiency of Qwen3-4B-Thinking-2507 model measured on one RTX 6000 Ada GPU. **(a)** End-to-end throughput vs. batch size (2048 in / 32768 out); dashed line = BF16 peak, dotted line = BF16's largest feasible batch. **(b)** Batch-1 prefill latency at 8k/16k/32k. **(c)** Generated tokens per problem on LiveCodeBench-v6, with the 32k cap shaded and the cap-hit rate per method.

5.2 Experiment Results

Tables 1, 2, and 3 summarize the main results. On general tasks, TaSQ achieves the strongest average performance across all models and leads on nearly every benchmark, substantially narrowing the gap to BF16. The improvement becomes more pronounced on long-CoT reasoning, where TaSQ preserves considerably more of the full-precision performance than existing VQ baselines. A similar trend appears in long-context retrieval: as the context length increases, TaSQ exhibits substantially less degradation. Interestingly, while NovaKV performs fairly well on general tasks, it suffers from severe performance degradation on long-context benchmarks. Appendix D further analyzes this trend at the attention-logit level beyond the calibration length.

6 Discussion

6.1 Efficiency Analysis

Serving efficiency. We compare the serving efficiency of TaSQ with the BF16 baseline and CQ, a minimal KV cache VQ baseline without additional runtime transformations. All three are evaluated within the same SGLang serving stack on a single RTX 6000 Ada GPU. As shown in Figure 4(a), TaSQ matches the throughput of CQ, indicating negligible additional serving overhead. Compared with BF16, it expands the KV cache pool from 235,257 to 2,935,184 tokens and increases the maximum batch size from 6 to 84. This raises peak throughput from 220.8 to 412.6 tokens/s, a $1.87\times$ improvement. Figure 4(b) shows that VQ encode overhead increases time-to-first-token (TTFT) by 10% to 14% for 8k to 32k prompts, but this one-time prefill cost is amortized over long generations.

Table 4: Component ablation of TaSQ on base Llama-3.1-8B. We report WikiText-2 perplexity at sequence length 2,048. Bold denotes the best quantized result.

Method	PPL ↓
FP16	6.2374
Full TaSQ	**8.4559**
w/o query-guided weighting	8.5172
w/o cross-head normalization	8.5103
w/o covariance-aware grouping	10.1731

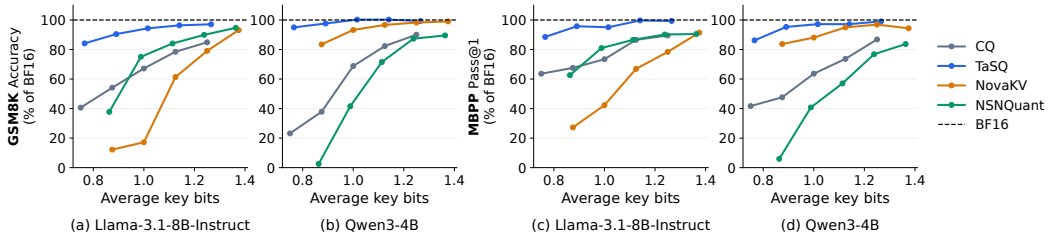


Figure 5: GSM8K and MBPP performance versus average key bits for Llama-3.1-8B-Instruct (a, c) and Qwen3-4B (b, d). Scores are normalized to each model's BF16 baseline (dashed, 100%); values remain in BF16.

Reasoning stability. Low-bit KV cache compression can destabilize reasoning models, causing repetitive or incoherent generation that continues until the length limit (Cheong et al., 2026). Figure 4(c) shows that TaSQ maintains output lengths and cap-hit rates close to BF16. Avoiding cap hits improves both accuracy and efficiency because unterminated generations mostly fail the task while consuming the full decoding budget. TaSQ therefore preserves reasoning stability while retaining the serving benefits of low-bit KV cache compression.

6.2 Ablation Study

Component ablation. To assess the contribution of individual method components, we measure perplexity (PPL) using Llama-3.1-8B on WikiText-2 with a context length of 2,048, while removing each component individually. We use the same calibration settings as in the main experiments and do not preserve any portion of the KV cache in full precision. The results in Table 4 show that while all components improve quantization quality, removing grouping results in the largest increase in perplexity, suggesting that grouping makes the largest contribution among the three components.

Evaluation under different bitwidths. We further compare the rate–accuracy trade-off on GSM8K and MBPP using Llama-3.1-8B-Instruct and Qwen3-4B, evaluating each method over five codebook sizes spanning comparable key rates. Since NovaKV uses scalar quantization for values, it cannot sweep the value rate into the sub-1-bit regime; we therefore keep values in BF16 for all methods and vary only the key rate. Exact configurations and bit accounting are provided in Appendix B. As shown in Figure 5, TaSQ consistently achieves the strongest rate–accuracy trade-off across both models, with the advantage becoming more pronounced at lower key rates.

7 Conclusion

We introduced TaSQ, which tailors the quantization target space for accurate and efficient KV cache VQ in the 1-bit regime. Our design incorporates query-guided weighting, cross-head normalization, and covariance-aware grouping while retaining the serving efficiency of conventional KV cache VQ methods. TaSQ consistently outperforms existing VQ baselines across general, reasoning, and long-context retrieval tasks while preserving generation stability. TaSQ supports up to $14\times$ larger batches and achieves up to $1.87\times$ higher throughput than the BF16 baseline in our SGLang implementation. Together, these results demonstrate the effectiveness of tailoring the VQ target space for ultra-low-bit KV cache compression.

AI USE STATEMENT

In this work, we used generative AI tools for method implementation and assistance with experimental setup and execution. We have not used generative AI tools for any other tasks requiring disclosure, and the remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for software debugging and for language editing and polishing of the manuscript. We have reviewed all AI-assisted work. LLM-generated code was reviewed, verified, and tested for correctness by two authors, and all reported results were checked against the corresponding experimental outputs. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 4 specifies the components and objectives of TaSQ, while Section 5.1 reports the models, datasets, calibration data, bit accounting, decoding policies, evaluation protocols, and hardware used in our experiments. The appendix provides additional derivations, algorithmic details, and ablation settings. We include the source code in the supplementary material, covering calibration and statistics collection, channel weighting and grouping, codebook fitting, SGLang integration, Triton kernels, and evaluation scripts.

REFERENCES

Marah Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, et al. Phi-4-reasoning technical report. *arXiv preprint arXiv:2504.21318*, 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Minsoo Cheong, Woosang Lim, Vincent-Daniel Yun, and Sungjoo Yoo. Kv-rescue: Recovering reasoning language model kv eviction loss via stepwise interleaving. *arXiv preprint arXiv:2608.15797*, 2026.

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

CodeParrot. Codeparrot clean. Hugging Face dataset, 2022. URL `https://huggingface.co/datasets/codeparrot/codeparrot-clean`.

Haojie Duanmu, Zhihang Yuan, Xiuhong Li, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. Skvq: Sliding-window key and value cache quantization for large language models. *arXiv preprint arXiv:2405.06219*, 2024.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Samuel Fernández-Menduiña, Amir Ziashahabi, Eduardo Pavez, Antonio Ortega, and Salman Avestimehr. Spend bits where queries look: Kv cache vector quantization with attention-preserving transforms. *arXiv preprint arXiv:2608.04074*, 2026.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Allen Gersho and Robert M Gray. *Vector quantization and signal compression*. Springer Science & Business Media, 2012.

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.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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

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

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *ArXiv*, abs/2403.07974, 2024. URL `https://api.semanticscholar.org/CorpusID:268379413`.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

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

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

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6: 87–100, 2024.

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. *arXiv preprint arXiv:2402.02750*, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Donghyun Son, Euntae Choi, and Sungjoo Yoo. Nsnquant: A double normalization approach for calibration-free low-bit vector quantization of kv cache. *Advances in Neural Information Processing Systems*, 38:43124–43159, 2026.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *Proceedings of machine learning research*, 235:48630, 2024.

Mart Van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635*, 2023.

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

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Think: Thinner key cache by query-driven pruning. In *International Conference on Learning Representations*, volume 2025, pp. 56691–56709, 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*, 2025.

Tianyi Zhang, Jonah Yi, Zhaozhuo Xu, and Anshumali Shrivastava. Kv cache is 1 bit per channel: Efficient large language model inference with coupled quantization. *Advances in Neural Information Processing Systems*, 37:3304–3331, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Related Work

LLM quantization initially focused on model weights. Scalar quantization (SQ), which quantizes individual weights independently, has been widely studied (Frantar et al., 2022; Lin et al., 2024; Xiao et al., 2023). At ultra-low bit rates, vector quantization (VQ) has gained attention for jointly encoding weight blocks and offering more favorable rate–distortion tradeoffs (Van Baalen et al., 2024; Egiazarian et al., 2024; Tseng et al., 2024).

KV cache quantization has emerged more recently as long contexts and larger serving batches shift the memory bottleneck from static weights to dynamically growing activations. Existing approaches span both SQ (Liu et al., 2024; Hooper et al., 2024; Duanmu et al., 2024) and VQ (Zhang et al., 2024; Son et al., 2026). Concurrent work NovaKV adopts a hybrid design, using dense transform-based VQ for post-RoPE keys and rotation-based SQ for values (Fernández-Menduiña et al., 2026). Our work belongs to the KV cache VQ line and studies the design of its native-basis pre-RoPE target space.

B Baseline Configurations and Bit Accounting

We report the effective KV cache rate in bits per element, accounting for both codebook indices and per-token metadata. All VQ methods use eight-channel groups, resulting in an index cost of $\log_2 K/8$ bits per element for a codebook with K centroids. All evaluated models have head dimension $d = 128$ and $H = 8$ except Phi4-14B-Reasoning-Plus model, which has $H = 10$ KV heads. We exclude static codebook storage and the full-precision prefix and recent windows, which are shared across methods.

For CQ and NSNQuant, we use their published configurations for the 1-bit regime (i.e., CQ-8c10b and NSNQuant-1b). Since NovaKV does not provide a configuration for this regime, we retain its original key VQ and rotation-based value SQ designs while setting the key codebook size to 1,024 centroids, matching CQ and TaSQ. Table 5 summarizes the resulting effective cache rates and their components.

Bitwidth sweep configurations. For the rate–accuracy sweep on GSM8K and MBPP in Figure 5, we vary the key codebook size over $K \in \{64, 128, 256, 512, 1024\}$ for CQ, NovaKV, and TaSQ. For NSNQuant, which incurs additional normalization metadata, we instead use $K \in \{32, 64, 128, 256, 512\}$ to cover a comparable range of effective key rates. Values are kept in BF16 for all methods because NovaKV uses scalar quantization for values and therefore cannot cover the sub-1-bit value-rate regime. Reported key rates include codebook indices and method-specific metadata, while excluding static codebook storage, full-precision cache windows, and implementation padding.

Table 5: Effective cache rate in bits per element. Per-token metadata costs are included in the reported rates.

Method	Channel	Representation	Rate
CQ	K	1,024-centroid VQ	$10/8 = 1.250$
	V	1,024-centroid VQ	$10/8 = 1.250$
NSNQuant	K	256-centroid VQ + normalization metadata	$8/8 + 0.238 = 1.238$
	V	256-centroid VQ + normalization metadata	$8/8 + 0.238 = 1.238$
NovaKV	K	1,024-centroid VQ + one 16-bit head-wise scale	$10/8 + 16/128 = 1.375$
	V	1-bit SQ + two 16-bit head-wise parameters	$1 + 32/128 = 1.250$
TaSQ	K	1,024-centroid VQ + one 16-bit cross-head scale	$10/8 + 16/(8 \cdot 128) = 1.266$
	V	1,024-centroid VQ	$10/8 = 1.250$

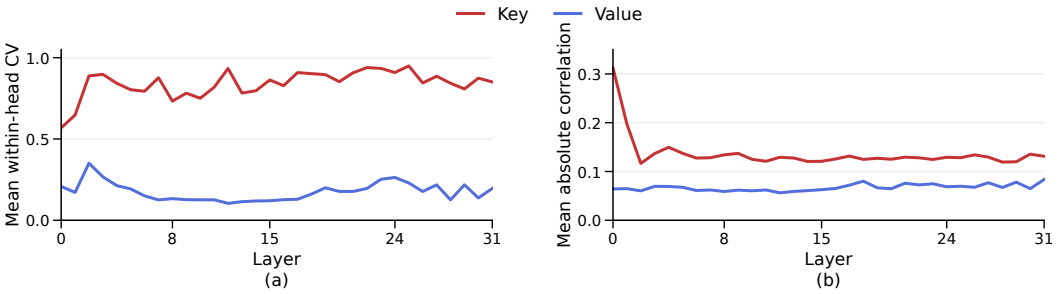


Figure 6: Key–value channel characteristics across layers of Llama-3.1-8B on the same 2,048-token GPQA segment. (a) Mean coefficient of variation (CV) of channel sensitivity; lower values indicate flatter sensitivity. (b) Mean absolute inter-channel Pearson correlation with the diagonal excluded. Both metrics are averaged over the 8 KV heads.

C Value Quantization Analysis

Key–value channel characteristics. We compare keys and values in terms of channel-wise sensitivity variation and inter-channel correlation using a 2,048-token GPQA segment with Llama-3.1-8B. We collect pre-RoPE keys, values, and their loss gradients across all 32 layers.

For $x \in \{k, v\}$, we measure the sensitivity of channel i in head h and layer l by its mean absolute loss gradient, and its variation by the coefficient of variation (CV):

$$a^x_{l,h,i} = \frac{1}{T}\sum_{p=1}^{T}\left|\frac{\partial L}{\partial x_{l,p,h,i}}\right|, \qquad \mathrm{CV}^x_l = \frac{1}{H}\sum_{h=1}^{H}\frac{\mathrm{Std}_i(a^x_{l,h,i})}{\mathrm{Mean}_i(a^x_{l,h,i})}.$$

Here, CV is computed within each head, with a lower CV indicating more uniform sensitivity across channels. As shown in Figure 6(a), values have substantially lower CV than keys across all layers (0.176 vs. 0.841 on average). This more uniform sensitivity makes the original value space better aligned with the uniform error metric of Euclidean VQ than the key space.

We also measure the mean absolute Pearson correlation between distinct channel pairs within each head. Figure 6(b) shows weaker inter-channel correlations for values across all layers (0.067 vs. 0.136 on average). Together, these results show that values exhibit both more uniform channel sensitivity and weaker inter-channel dependencies than keys, motivating our focus on tailoring the quantization space for keys. We leave tailoring the quantization space for values to future work.

Explored value-side alternatives. Our final method leaves the value path unchanged and applies plain contiguous VQ. As an exploratory ablation, we tested whether the key-side weighting, normalization, and grouping choices also benefit value quantization. Following the derivation of query-guided key weighting in Section 4.1, we first derive an analogous channel-wise sensitivity for values. Let $\delta v_{p,h} = v_{p,h} - \hat{v}_{p,h}$ denote the value reconstruction error at position p and KV head h. The contribution of this error to the attention output at query position t is

$$\Delta o_{t,p,h} = a_{t,p,h} W_{O,h} \delta v_{p,h},$$

where $a_{t,p,h}$ is the attention weight and $W_{O,h}$ denotes the block of the output projection corresponding to head h. Its squared norm is

$$\|\Delta o_{t,p,h}\|_2^2 = \delta v_{p,h}^\top \left(a_{t,p,h}^2 W_{O,h}^\top W_{O,h}\right) \delta v_{p,h}.$$

Since $a^2_{t,p,h}$ scales all value channels equally, the relative importance of value reconstruction errors is determined by $W_{O,h}^\top W_{O,h}$. Following the diagonal approximation used for keys, we therefore use $\mathrm{diag}(W_{O,h}^\top W_{O,h})$ to construct channel weights for values.

Based on the derived weights, we evaluate the combined application of value-side channel weighting, cross-head normalization, and covariance-aware grouping on Llama-3.1-8B while keeping the key path fixed. We otherwise follow the main calibration and VQ settings, except that value grouping starts from individual channels since values are not subject to RoPE-pair constraints. Table 6

Table 6: Value-side ablation on Llama-3.1-8B: WikiText-2 perplexity at sequence length 2,048 (lower is better). The TaSQ key path is identical in both rows.

Value quantization	PPL ↓
TaSQ	8.45594
TaSQ + value-side transformations	**8.42058**

Algorithm 1 RoPE-pair-constrained hierarchical partitioning

Require: covariance Σ_h^{wn}; group size g
1: $U \leftarrow \{\{2j-1, 2j\} : j = 1, \ldots, d/2\}$
2: **for** $r = 1$ **to** $\log_2(g/2)$ **do**
3: assign each edge (u, v) between distinct units in U the cost $c_h(u \cup v)$
4: $M \leftarrow$ minimum-weight perfect matching on U
5: $U \leftarrow \{u \cup v : (u, v) \in M\}$
6: **end for**
7: **return** U and the corresponding grouping layout

shows that these transforms reduce WikiText-2 perplexity from 8.45594 to 8.42058 (0.42%). Given the modest gain and the additional bits that scales require, we retain standard VQ with contiguous channel groups for values.

D Method Details and Analysis

Hierarchical matching for covariance-aware grouping. Section 4.3 formulates covariance-aware grouping as a partitioning problem that minimizes the total group cost subject to equal-size groups and RoPE-pair preservation. Let $n = d/2$ denote the number of atomic RoPE pairs and $m = g/2$ the number of pairs per group. The number of possible partitions satisfying these constraints is

$$\frac{n!}{(m!)^{n/m}(n/m)!},$$

making exhaustive search infeasible. We therefore approximately solve this problem using a hierarchical procedure based on minimum-weight perfect matching, as detailed in Algorithm 1.

Each round finds the exact minimum-cost pairing of its current groups. Since earlier merges constrain the choices available in subsequent rounds, the hierarchical procedure does not guarantee a globally optimal final partition.

Cross-head shared scale. Section 4.2 derives the metadata cost of cross-head normalization. With a b_s-bit scale and head dimension d, head-wise normalization requires b_s/d bits per channel, whereas sharing the scale across H heads reduces this cost to $b_s/(Hd)$. Under a fixed bit budget, these savings can be reassigned to codebook indices, providing more centroids to approximate each channel group and thereby reducing VQ distortion.

		Llama-3.1-8B			Qwen3-4B		
scale / codebook	bits/elem	mean	min	max	mean	min	max
per-head, $K = 1024$	1.3750	0.004483	0.002882	0.006953	0.019794	0.006732	0.030261
pooled, $K = 1024$	1.2656	0.004564	0.002965	0.007037	0.019688	0.006754	0.030014
pooled, $K = 2048$	1.3906	0.003460	0.002245	0.005313	0.014763	0.005045	0.022639

Table 7: Relative SSE with separately fitted codebooks, across all layers. Bits per channel include fp16 scale metadata with $g = 8$, $d = 128$, and $H = 8$.

At equal K, pooling is distortion-neutral: relative SSE stays within $\pm 4\%$ of per-head on every layer of both models, while saving 0.109 bits per channel. That is just short of the $1/8$ bit a doubling of K costs, so pooled $K = 2048$ spends 0.016 bits per channel more than per-head $K = 1024$; in

exchange, relative SSE drops by at least 20% in all 68 layers measured. With an 8-bit token scale and a shared block-level fp32 scale, pooling saves 0.0547 bits per channel for $H = 8$ and $d = 128$.

Generalization beyond the calibration length. Because RoPE induces position-dependent changes in the key distribution, methods fitted to post-RoPE activations may be sensitive to positions beyond those observed during calibration. To examine this effect, we evaluate all four methods on eight 16K WikiText-2 sequences per model, using the same calibration setting as in the main experiments. Figure 7 reports attention-logit NMSE over bins of 128 key positions, computed from causally valid query–key pairs across layers and heads.

For the pre-RoPE VQ methods, CQ and TaSQ, the error increases only gradually beyond the calibration range, with TaSQ maintaining the lowest error across positions. NSNQuant, despite operating on post-RoPE keys, also remains relatively stable, due to its calibration-free codebook design. In contrast, NovaKV exhibits a sharp increase in error beyond the calibration range. Its transforms and codebooks are fitted directly to post-RoPE activations from the calibration data, making them more sensitive to position-dependent distribution shifts outside the observed range. This trend is consistent with the long-context retrieval results in Table 3, where NovaKV degrades sharply as context length increases, while TaSQ maintains substantially stronger performance.

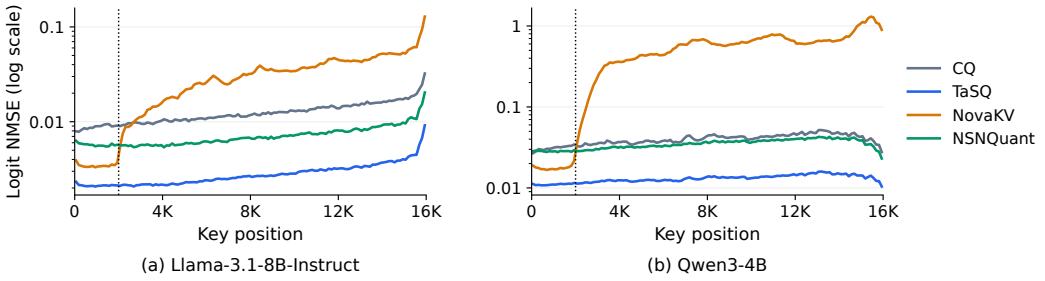


Figure 7: Attention-logit NMSE (log scale) versus key position for Llama-3.1-8B-Instruct (a) and Qwen3-4B (b). Dotted lines mark the 2048 calibration length.

Justification for the diagonal approximation in query-guided channel weighting. Section 4.1 uses $W_h = \mathrm{diag}(\tilde{A}_h)$ and transforms keys with $W_h^{1/2}$ rather than the dense $\tilde{A}_h^{1/2}$. We justify this choice by comparing decode cost and evaluation error.

Let $u_{p,h} = A_h k_{p,h}$ denote a transformed key and $\bar{q}_{t,h}$ a post-RoPE query. At decoding, the key must be restored by A_h^{-1} before applying its position-dependent RoPE rotation:

$$\bar{q}_{t,h}^{\top} R_p k_{p,h} = \bar{q}_{t,h}^{\top} R_p A_h^{-1} u_{p,h}. \tag{2}$$

When A_h is diagonal, this restoration can be absorbed into the codebooks and adds no runtime transform. In contrast, a dense A_h^{-1} requires a $d \times d$ multiplication for every cached key. It cannot be merged into the query projection because R_p depends on the key position. Table 8 shows that this additional cost grows from 36% of attention time at 1k tokens to 103% at 16k.

Context	Latency (ms/step)		Overhead
(tokens)	Attention	Dense Restoration	(%)
1 024	1.55	0.55	+36%
4 096	2.11	1.23	+58%
16 384	4.14	4.27	+103%

Table 8: Cost of dense key restoration for Qwen3-4B-Thinking-2507 on one RTX 3090 at batch size 1. Restoration is measured as a standalone BF16 GEMM over reconstructed keys, excluding VQ decoding; overhead is relative to attention latency.

To test its effect on quantization quality, we fit two variants from the same calibration data using either $\mathrm{diag}(\tilde{A}_h)^{1/2}$ or $\tilde{A}_h^{1/2}$, while holding all other settings fixed. We then measure relative attention-logit error on the calibration corpus and GSM8K, using $\sum_{t,p}(\bar{q}_{t,h}^{\top} R_p \delta_{p,h})^2 / \sum_{t,p}(\bar{q}_{t,h}^{\top} R_p k_{p,h})^2$.

Key Transform	Relative Attention-Logit Error	
	Calibration Corpus	GSM8K
$\mathrm{diag}(\tilde{A}_h)^{1/2}$ (TaSQ)	0.00287	**0.00234**
$\tilde{A}_h^{1/2}$	**0.00264**	0.00933

Table 9: Relative attention-logit error for Llama-3.1-8B-Instruct, averaged over 32 layers. Both variants use the same calibration data and $K = 1024$ codebooks.

Dense weighting slightly lowers error on the calibration corpus but yields about $4\times$ higher error on GSM8K. Its off-diagonal channel couplings do not transfer in this experiment. The diagonal approximation therefore preserves efficient decoding while achieving lower error on GSM8K.