TORQUE: OPTIMIZING WHAT (NOT) TO QUANTIZE BEFORE AND AFTER ROTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Uniform random rotations are an effective preprocessing step for quantization: they make normalized coordinate distributions approximately Gaussian, enabling the use of codebooks optimized offline. We introduce TORQUE, a framework that improves on previous quantization works that use random rotations by jointly optimizing how many and which coordinates to preserve at high precision both before and after rotation, under a fixed overall expected bit budget.

Intuitively, before rotation, preserving large input coordinates at high precision can reduce overall error by preventing the rotation from spreading their values across many coordinates. Likewise, after rotation, preserving a small fraction of the largest-magnitude coordinates at high precision allows the remaining values to be quantized more accurately using codebooks optimized offline for the resulting truncated Gaussian distribution.

We derive a quantization error upper bound and prove that top-k pre-rotation retention minimizes it for each k. This reduces the search over coordinate subsets to an optimization over k, enabling a fast optimizer that uses offline codebooks and parallel parameter selection for practical implementation. We demonstrate an improved tradeoff between reconstruction accuracy and storage cost through numerical evaluation under the Gaussian model and experiments on nearest-neighbor retrieval, KV-cache compression, and activation compression.

1 INTRODUCTION

Random-rotation-based quantization reduces the memory, communication, and computation costs of machine learning tasks, with applications in distributed training (Vargaftik et al., 2022; Suresh et al., 2017; Vargaftik et al., 2021; Ben-Basat et al., 2024) and inference (Zandieh et al., 2026; Chee et al., 2023; Tseng et al., 2024; Ashkboos et al., 2024; Liu et al., 2025; Malinovskii et al., 2025), and approximate nearest-neighbor retrieval (Gao & Long, 2024). Random rotations regularize the coordinate distribution: under a uniform random (Haar) rotation, every fixed-norm input induces the same spherical distribution, whose fixed-size marginals approach a Gaussian. This allows the use of precomputed codebooks optimized for that distribution with input-independent accuracy guarantees, while randomized Hadamard transforms provide efficient practical implementations (Vargaftik et al., 2022; Yang et al., 2025; Panferov et al., 2026). Production frameworks also support this approach: vLLM includes a backend with Hadamard rotation, and its LLM Compressor supports rotation-based transforms (vLLM, 2026; LLM Compressor, 2026).

We identify two potential sources for improvement in quantization based on random rotations. First, a rotation can obscure useful structure in the original vector. For example, when a few coordinates have particularly large magnitudes (*outliers*), preserving them at higher precision before the rotation can be more effective than spreading their values across the rotated vector. Second, the rotated vector can yield outliers that benefit from separate, higher-precision encoding, while the remaining values (the *inliers*) are passed to a lower-bit quantizer. Importantly, this allows the inliers to be quantized over a narrower range, increasing accuracy. We jointly optimize this outlier selection before and after the rotation to leverage both opportunities under a fixed bit budget. Outlier retention was previously considered before and after rotation *in isolation* and *without such an optimization*.

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.

Before rotation: preserving structure in the input. Special handling of outliers is common even in methods that use no rotation. LLM.int8() isolates outlier feature dimensions for 16-bit multiplication while processing the remaining features in 8 bits (Dettmers et al., 2022); this mechanism is available in Hugging Face's bitsandbytes (bitsandbytes contributors, 2026). SpQR preserves sensitive individual weights at higher precision, and OWQ preserves sensitive weight columns (Dettmers et al., 2024; Lee et al., 2024). AWQ also motivates protecting salient weights, but implements that protection through activation-aware channel scaling rather than a retained high-precision subset (Lin et al., 2024).

After rotation: quantizing a bounded Gaussian distribution. Retention of post-rotation outliers has previously been considered by QUIC-FL (Ben-Basat et al., 2024). QUIC-FL rotates the input, transmits values outside a central interval separately, and optimizes an unbiased stochastic quantizer for the bounded inlier distribution. There, rotation and outlier retention are complementary: the rotation supplies a predictable distribution, and retaining outliers separately allows finer quantization of the remaining coordinates. Overall, this results in increased accuracy.

The central question is *how many and which values to retain at each stage*. While previous works considered outlier retention at different stages, they did not jointly allocate one vector's bit budget across pre-rotation retention, post-rotation retention, and inlier quantization. The decisions are coupled: retaining values reduces the error but costs value and index bits that could otherwise improve the inliers; pre-rotation retention reduces the norm of the remaining vector before normalization.

We introduce TORQUE (*Two-stage Outlier-retention Rotated QUantization FramEwork*), a framework that models all these choices as a single optimization problem. Given a total expected bit budget, it determines the pre-rotation and post-rotation outlier retention policies, and the inlier bit-budget and codebook, charging both retained values and their positions. Importantly, the inlier codebooks are designed offline for the resulting induced (truncated Gaussian) distribution.

TORQUE applies to scalar and vector quantizers. Moreover, TORQUE preserves unbiasedness when the inlier reconstruction is unbiased. Namely, unbiased estimation of the inliers, exact retention of outliers, and linear recombination through the inverse rotation yield an unbiased estimate.

Contributions.

1. **Joint optimization formulation and framework.** We identify outlier retention before and after rotation as coupled decisions and introduce TORQUE, a framework that jointly optimizes them under a shared bit budget. Our formulation accounts for the costs of retained values, their indices, and quantizing the remaining coordinates.
2. **Theoretical structure and fast online solution.** We derive an NMSE upper bound and prove that retaining the largest-magnitude input coordinates before the rotation minimizes this bound. This reduces the search over coordinate subsets to optimizing their size. We combine this with offline codebook design and parallel candidate evaluation to enable fast online parameter selection.
3. **Evaluation across quantizers and ML workloads.** We numerically quantify worst-case error reductions for scalar and vector quantizers and evaluate on nearest-neighbor retrieval, KV-cache compression, and activation compression. Our results demonstrate that TORQUE consistently improves the tradeoff between reconstruction accuracy and bit-budget for rotation-based scalar and vector quantization schemes.

2 THE TORQUE FRAMEWORK

Figure 1 illustrates TORQUE. Given the input and a target bit budget, the joint optimization selects the retained outliers before and after rotation, and the quantizer hyperparameters for the remaining (inlier) entries. We first explain the general workflow and then define the optimization problem.

Specifically, we first describe TORQUE with recommended design choices that trade off compression error and computational complexity. Appendix B discusses alternative implementation choices.

2.1 ENCODING AND DECODING

Let $k \in \{0, 1, \ldots, d\}$, $c \in C \subset (0, \infty]$ and $s \in S \subset [0, \infty)$ be parameters that our joint optimization (§2.3) chooses based on the nonzero input $x \in \mathbb{R}^d$ and the bit-budget $b \geq 0$. Here, C and S are

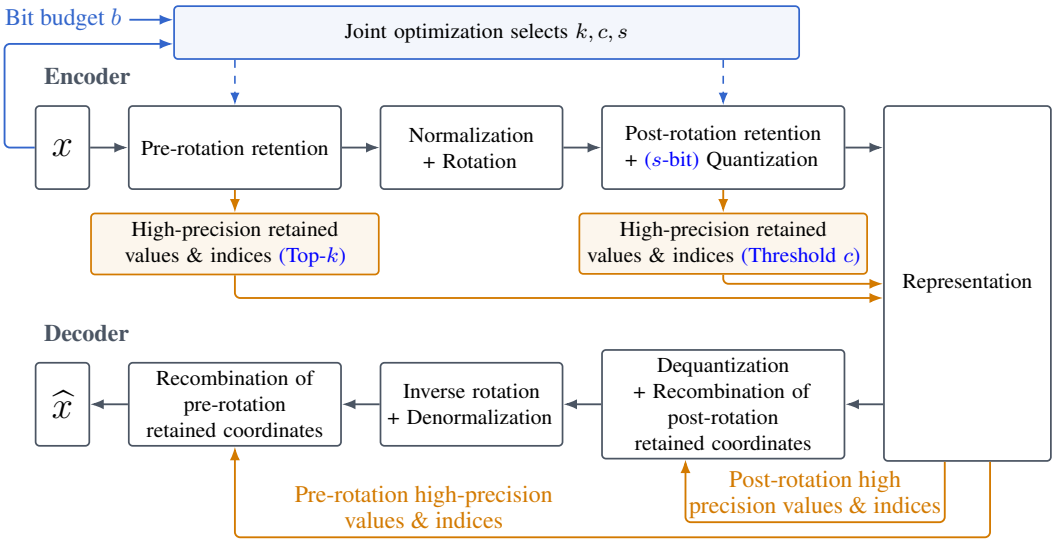


Figure 1: Illustration of the TORQUE framework. Given an input vector $x \in \mathbb{R}^d$ and a total budget of db bits, the joint optimization selects the number k of largest-magnitude coordinates to retain at high precision before rotation, the retention threshold c after rotation, and the bit budget s with a matching offline precomputed codebook $Q_{c,s}$ for quantizing the remaining entries.

finite sets of supported thresholds and bit allocations. As detailed in §2.2, for each (c, s), we use an (offline) precomputed optimized codebook $Q_{c,s}$.

Encoding. First, the encoder stores the k largest-magnitude input coordinates, along with their positions, and sets those positions in the input to zero. Importantly, the input keeps its original dimension throughout encoding, enabling fast in-place rotation. In practice, since only a small number of coordinates are expected to be stored separately, the additional memory overhead is small.

Choosing the k largest-magnitude coordinates minimizes the error bound for a given k: after normalization and a uniform random rotation,[1] every input induces the same spherical distribution and hence the same expected quantization error. Therefore, as we prove in Appendix A, the optimization only needs to search over how many coordinates to retain.

After normalization and rotation, values whose magnitude exceeds the threshold c are stored separately with high precision. For vector quantization, we retain the q-sized block if its norm exceeds c. The values that are not retained, i.e., the inliers, are then quantized using the precomputed codebook $Q_{c,s}$ which is optimized for the induced truncated Gaussian distribution.

Finally, the encoded representation includes c, s, the retained values and positions from both retention stages, the quantized inlier encoding, and the normalization factor.

Decoding. The decoder starts by reconstructing the inliers and reinserting the values retained after rotation. It then applies the inverse rotation and reverses the normalization. Finally, it overrides the entries in the reconstructed vector with the values retained before rotation, using their original positions. Importantly, if the quantization is unbiased, the decoding preserves unbiasedness.

[1]For theoretical analysis, we assume that the encoder and decoder share a uniform random rotation, also known as the Haar rotation, while randomized Hadamard transforms provide efficient practical implementations (Vargaftik et al., 2022; Ben-Basat et al., 2024; Yang et al., 2025; Panferov et al., 2026). Indeed, recent work has demonstrated that RHTs can approximate Haar rotations with provable guarantees (Ben-Basat et al., 2026; Zilca & Mendelson, 2026; Feng et al., 2026).

2.2 FORMULATION

Error measure. Throughout the paper, we measure reconstruction error using the vector normalized mean squared error (NMSE). For a nonzero input $x \in \mathbb{R}^d$ and its reconstruction $\widehat{x}$, this is

$$\text{NMSE} = \frac{\mathbb{E}\|\widehat{x} - x\|_2^2}{\|x\|_2^2}, \tag{1}$$

where the expectation is over the rotation and any randomness in the quantizer.

Input normalization and retained coordinates. For $k \in \{0, \ldots, d\}$, let $x_{-k} \in \mathbb{R}^d$ be the vector obtained by setting the k largest-magnitude coordinates to zero. We also use $x_k = x - x_{-k}$ to denote the vector with the largest-magnitude coordinates and zeros elsewhere.

Let $\rho_k = \frac{\|x_{-k}\|_2^2}{\|x\|_2^2}$ be the fraction of the squared input norm remaining after retention. If $\|x_{-k}\|_2 = 0$, all remaining coordinates are zero. The retained values already specify the entire input, so we skip rotation and quantization. This case has zero error and pays only the cost of pre-rotation retention.

When $\|x_{-k}\|_2 > 0$, we normalize it as $z_k = x_{-k} \cdot \sqrt{d}/\|x_{-k}\|_2 \in \mathbb{R}^d$, which satisfies $\|z_k\|_2^2 = d$.

Rotation and quantization. Let R be the rotation and write $U = Rz_k$. The rotation makes the expected squared value of each coordinate equal to one.

For scalar quantization, we retain each rotated coordinate whose magnitude exceeds a threshold c. Each remaining coordinate lies in $[-c, c]$ and is encoded with a scalar quantizer.

The extension to vector quantization is direct. We partition U into blocks of q coordinates, where q divides d, and retain a whole block U_j when $\|U_j\|_2 > c$. Each remaining block is encoded with a vector quantizer. Observe that the case $q = 1$ recovers scalar quantization, and $c = \infty$ retains no rotated values in either case. Pre-rotation retention still selects individual coordinates.

Offline codebook construction. Before encoding any input, we choose a set C of candidate thresholds and a set S of candidate bit allocations. Here $s \in S$ is the number of bits per inlier coordinate. For each pair $(c, s) \in C \times S$, we design a quantizer $Q_{c,s}$ from the chosen scalar or vector family. The threshold determines the range of values the codebook represents, while s determines how many bits are available to encode them.

Specifically, for a scalar quantizer, we optimize the codebook for a standard Gaussian conditioned to lie in $[-c, c]$. For a vector quantizer, we use a Gaussian block $G \sim N(0, I_q)$ conditioned on $\|G\|_2 \leq c$. We focus on centroid codebooks for these truncated Gaussian distributions, but our framework also supports any other codebook design.

Importantly, for each pair (c, s), we compute the codebook $Q_{c,s}$ *once*, *offline*, and reuse it across input dimensions. We also precompute the expected error of the complete reconstruction, including post-rotation retention, to allow a fast search during optimization.

We first consider scalar quantization. Let $G \sim N(0, 1)$. Values with $|G| > c$ are retained exactly, while the remaining values are reconstructed using $Q_{c,s}$. The Gaussian-model error is therefore

$$\epsilon(c, s) = \mathbb{E}\left[(Q_{c,s}(G) - G)^2 \cdot \mathbf{1}_{\{|G| \leq c\}}\right].$$

Namely, only inliers contribute error.

For vector quantization, we apply the same construction to blocks of q coordinates. We now take $G \sim N(0, I_q)$ and retain blocks whose norm exceeds c. The normalized error becomes

$$\epsilon(c, s) = \frac{1}{q}\mathbb{E}\left[\|Q_{c,s}(G) - G\|_2^2 \cdot \mathbf{1}_{\{\|G\|_2 \leq c\}}\right]. \tag{2}$$

We divide by q because $\mathbb{E}\|G\|_2^2 = q$. Both expectations include any randomness in the quantizer.

At runtime, the joint optimization selects a codebook together with the 2-stage retention parameters.

Storing retained values and their positions. Each retained value requires bits for the value itself and for its position. We use fixed-width indices for the positions. Let $v_{\text{pre}}, v_{\text{post}}$ denote the value costs before and after rotation, and let $p_{\text{pre}}, p_{\text{post}}$ denote the corresponding index costs. The total costs per retained scalar are

$$L_{\text{pre}} = v_{\text{pre}} + p_{\text{pre}}, \qquad L_{\text{post}} = v_{\text{post}} + p_{\text{post}}. \tag{3}$$

The expected bit budget. Let b be the target number of bits per input coordinate, giving a total budget of db. We first consider scalar quantization of a nonzero remaining vector. Let $p_d(c)$ be the expected fraction of coordinates retained after rotation. The expected number of stored bits is

$$\mathcal{B}(k, c, s) = kL_{\text{pre}} + d\big[p_d(c)L_{\text{post}} + (1 - p_d(c))s\big]. \tag{4}$$

The first term counts the values and positions retained before rotation. Inside the brackets, the first term pays for retention after rotation and the second pays for the inlier codes.

When $\rho_k = 0$, we instead set $\mathcal{B} = kL_{\text{pre}}$ and the objective to zero. Any required metadata costs are added before checking feasibility.

For vector quantization, we retain a block U_j when $\|U_j\|_2 > c$. We therefore replace the scalar retention probability $p_d(c) = p_{d,1}(c)$ with $p_{d,q}(c) = \mathbb{P}(\|U_j\|_2 > c)$. Each coordinate is retained whenever its block is retained. Thus, $p_{d,q}(c)$ is also the expected fraction of coordinates retained. Each retained block needs only one index, so its index cost per coordinate, p_{post}, is the number of bits in that index divided by q. With these changes, Equation (4) also applies to vector quantization.

The number of values retained after rotation varies with the rotation. Thus, $\mathcal{B}$ is the expected number of bits used to store retained values, their positions, and inlier codes, averaged over the rotation. The complete representation also include metadata (e.g., the normalization factor and quantizer identifiers). Since the target budget covers the complete representation, we add the metadata costs to $\mathcal{B}$ before checking the budget constraint.

2.3 Joint optimization and solution

Reconstruction error. The decoder first uses the quantizer to estimate inlier coordinates and overrides the retained position with their values. Let $\widehat{U}$ denote this estimate. Inverse rotation gives the approximation of z_k:

$$\widehat{z}_k = R^\top \widehat{U}.$$

A rotation preserves squared distances. Since $\|z_k\|_2^2 = d$, its NMSE is

$$\epsilon_d(c, s) = \frac{1}{d}\mathbb{E}\|\widehat{z}_k - z_k\|_2^2 = \frac{1}{d}\mathbb{E}\|\widehat{U} - U\|_2^2.$$

The distribution of U is the same for every z_k. Thus, for the selected quantizer family and dimension d, $\epsilon_d(c, s)$ can be precomputed *offline* for each pair (c, s). The vector U is uniform on the sphere of radius $\sqrt{d}$, and each coordinate has a scaled, shifted Beta distribution (Vargaftik et al., 2021). In practice, we approximate its fixed-size block marginals by Gaussians and use $\epsilon(c, s)$ as an approximation to $\epsilon_d(c, s)$.

To restore the scale, we multiply by $\|x_{-k}\|_2/\sqrt{d}$. Before overwriting the retained input positions, this gives the intermediate reconstruction

$$\widetilde{x} = x_k + \frac{\|x_{-k}\|_2}{\sqrt{d}}\widehat{z}_k, \qquad x = x_k + \frac{\|x_{-k}\|_2}{\sqrt{d}}z_k.$$

Under exact retention, x_k appears in both expressions and cancels when we subtract them. Therefore,

$$\frac{\mathbb{E}\|\widetilde{x} - x\|_2^2}{\|x\|_2^2} = \rho_k\, \epsilon_d(c, s).$$

At the retained input positions, z_k is zero, so any nonzero reconstructed value there is noise. The decoder replaces this noise with zero before adding the retained values. This can only decrease the squared error. The final reconstruction $\widehat{x}$ therefore satisfies

$$\text{NMSE} = \frac{\mathbb{E}\|\widehat{x} - x\|_2^2}{\|x\|_2^2} \le \frac{\mathbb{E}\|\widetilde{x} - x\|_2^2}{\|x\|_2^2} = \rho_k\, \epsilon_d(c, s).$$

Selecting the parameters. We use the Gaussian approximation of this bound to compare candidates and choose the feasible combination with the smallest modeled error:

$$\boxed{(k^*, c^*, s^*) \in \underset{k\in\{0,\dots,d\},\ c\in C,\ s\in S:\ \mathcal{B}(k,c,s)\le db}{\arg\min} \rho_k\, \epsilon(c,s).} \tag{5}$$

The search uses integer values of k, and the dimension stays d for every candidate. When the remaining vector is zero, we use the exact reconstruction from retained values described above.

The choices affect each other. Retaining more input coordinates reduces the squared norm of the remaining vector, but leaves fewer bits for quantizing it. Similarly, lowering the post-rotation threshold narrows the inlier distribution, but spends more bits on high-precision values.

We include $k = 0$ and $c = \infty$ among the candidates. This allows retention only after rotation, as well as quantization without retention at either stage. Thus, the selected Gaussian-model objective cannot exceed that of any feasible baseline included in the candidate set.

Solution complexity. As we prove in Appendix A, pre-rotation retention reduces to choosing the number of retained coordinates. We present two alternative implementation options.

For fixed (c, s), ρ_k is nonincreasing in k, while $\epsilon(c, s) \ge 0$ is independent of k. Thus, the largest feasible k minimizes their product. Let $T(c, s) = d[p_d(c)L_{\text{post}} + (1 - p_d(c))s]$. Thus, for each (c, s) we obtain $k_{\max}(c, s) = \min\{d, \lfloor (db - T(c, s))/L_{\text{pre}} \rfloor\}$, provided $k_{\max}(c, s) \ge 0$. Otherwise, the pair c, s is infeasible. We sort the squared coordinates once, in $O(d \log d)$ time, and compute their cumulative sums in $O(d)$ time. Evaluating $\rho_{k_{\max}(c,s)}\epsilon(c, s)$ for a given (c, s) then takes $O(1)$ time, giving total parameter-selection time $O(d \log d + |C||S|)$. Moreover, the evaluation of each (c, s) candidate is independent and can be parallelized on a GPU.

For the second implementation, let $\mathcal{K} = \{k_{\max}(c, s) \mid c \in C, s \in S\}$ be the set of all k values that are of interest. We run the Multiple Selection algorithm that allows computing the $|\mathcal{K}|$ quantiles of $|x|$ in $O(d \log |\mathcal{K}|)$ time. Accumulating squared-coordinate sums during partitioning gives each required $\|x_{-k}\|_2^2$ and hence ρ_k, with ties split to retain exactly k coordinates. The total parameter selection time is thus $O(d \log(|C||S|) + |C||S|)$.

Combining the two options, we obtain a runtime bound of $O(|C||S| + d \cdot \min\{\log d, \log(|C||S|)\})$.

Small candidate sets are natural in practice. The implementation typically supports only a few bit allocations, keeping S small. We can choose and refine the thresholds in C offline until additional candidates offer little improvement. This balances accuracy against codebook storage and search time, while keeping codebook construction offline and allowing retention to adapt to each input.

For the rotation, the practical choice is to apply a randomized Hadamard transform to z_k at $O(d \log d)$ time, and the decoder can apply its inverse. This option uses the same retention rules and precomputed codebooks. Moreover, using Hadamard compositions or dithering offer $O(d \log d)$ time while providing theoretical guarantees that asymptotically match those of a uniform random rotation (Ben-Basat et al., 2026; Zilca & Mendelson, 2026; Feng et al., 2026).

Overall, the complexity remains largely determined by the inlier quantizer and its rotation. TORQUE reuses these operations, adding parameter selection and $O(d)$ work for thresholding and recombination. For small, fixed sets C, S, this additional work is $O(d \log d)$. Thus, even when the inlier quantizer uses the fast $O(d \log d)$ rotation, TORQUE preserves its asymptotic complexity.

3 Experiments

The application experiments compare the scalar quantizers EDEN (Vargaftik et al., 2022), RaBitQ (Gao & Long, 2024; Gao et al., 2025), and TurboQuant (Zandieh et al., 2026), with and without TORQUE. Section 3.4 also evaluates the vector quantizer HIGGS (Malinovskii et al., 2025) under the Gaussian model. The main figures use unbiased scalar reconstruction and Gaussian reciprocal reconstruction for HIGGS. Biased variants appear in Appendix C.

Retained values cost $v = 16$ bits each and indices cost $p = 8$ bits each. For scalar applications, we divide vectors into consecutive chunks of at most 256 coordinates. A vector block of q coordinates shares one index, so its retention costs $16q + p$ bits and supports at most 256 blocks per rotation.

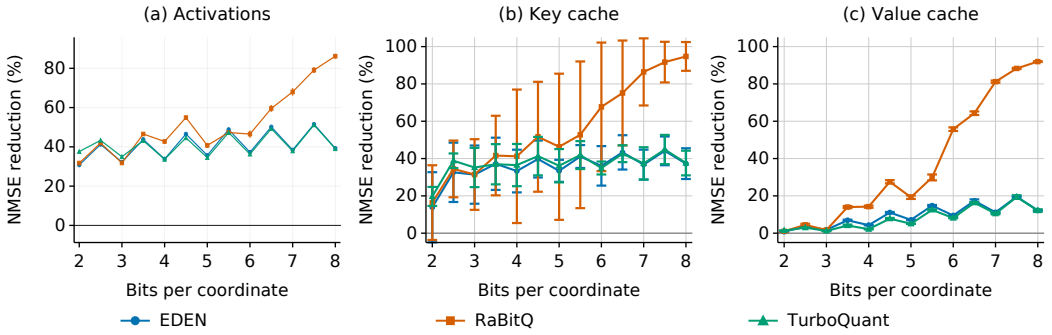


Figure 2: Qwen2.5-0.5B TORQUE improvement: (a) activations; (b) key caches; (c) value caches.

3.1 Activation and KV-cache compression

We evaluate Qwen2.5-0.5B (Qwen Team, 2024) on the WikiText-2 test split (Merity et al., 2017). Here, we compare the unbiased variants of the above quantizers, while the biased variants are evaluated in Appendix C.1.

Activations. We run the model on eight text segments of 256 tokens each and choose eight tokens from each segment. For these 64 tokens, we record four activation vectors within each of the model's 24 decoder blocks: the input to the query/key/value projections, the input to the attention output projection, the input to the MLP gate/up projections, and the input to the MLP down projection. We quantize each recorded vector in consecutive chunks of 256 coordinates. If 128 coordinates remain at the end, we quantize them as a separate chunk of size 128. For TORQUE, each chunk uses both retention stages.

As shown in Figure 2(a), TORQUE reduces the NMSE by 30.8–51.5% for EDEN, 31.8–86.2% for RaBitQ, and 33.6–51.2% for TurboQuant. Interestingly, for non-integer values of b, our improvement is larger than for integer ones. This is because it allows TORQUE to use an integer number of bits when quantizing inliers and use the residual bits for the retained values. For the raw inlier quantizers, a fractional budget means mixed-precision quantization, leading to a less favorable bit-budget to accuracy tradeoff as shown in EDEN (Vargaftik et al., 2022).

KV caches. We run the model on 16 nonoverlapping text segments of 256 tokens each. For each segment, we process the first 255 tokens to construct the key and value caches in all 24 decoder blocks. Each decoder block has two key/value heads, each with 64 coordinates, giving 128 key coordinates and 128 value coordinates per token. We quantize each head's key and value vectors separately, using both retention stages for TORQUE.

As shown in Figure 2(b) and (c), TORQUE reduces mean key and value NMSE at every tested budget. Key NMSE decreases by 13.9–44.1% for EDEN, 16.4–94.7% for RaBitQ, and 19.7–44.8% for TurboQuant. The corresponding value NMSE reductions are 0.8–19.6%, 1–92%, and 1.2–19.4%.

3.2 Nearest-neighbor retrieval

Figure 3 uses the complete GloVe-200 (Pennington et al., 2014) index of 1,183,514 vectors and all 10,000 queries, with cosine similarity. We reconstruct the quantized database vectors and measure NMSE and Recall@1: the fraction of queries for which the retrieved nearest neighbor agrees with the full-precision search.

Figure 3(a) shows the NMSE reduction, (b) the change in recall, and (c) Recall@1. Appendix C.2 includes the GIST (Jegou et al., 2010) dataset and biased retrieval comparisons.

At the same bit budget, TORQUE lowers NMSE for all three quantizers. Across the 13 plotted budgets $b = 2, 2.5, \ldots, 8$, the mean paired NMSE reductions are up to 20.9% for EDEN, 81.5% for RaBitQ, and 20.3% for TurboQuant, respectively. Recall@1 also improves, with gains of up to 1.5 percentage points.

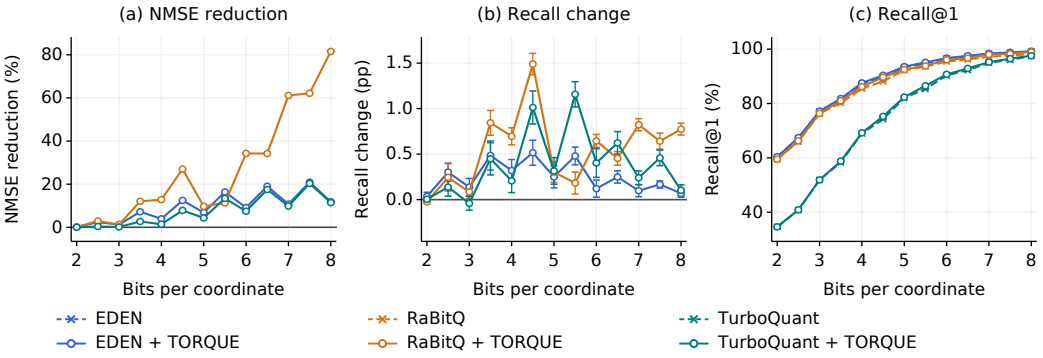
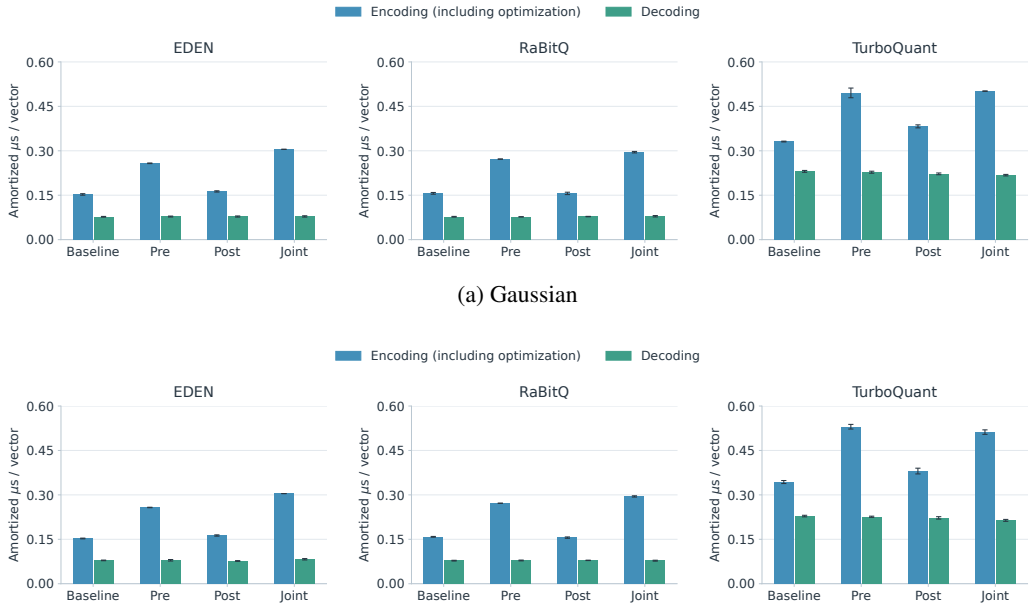


Figure 3: GloVe retrieval: (a) NMSE reduction, (b) change in recall, and (c) Recall@1.



(a) Gaussian

(b) Student-t_3

Figure 4: CUDA encoding and decoding time on an NVIDIA RTX 5090 for $d = 256$, batch size 128, and a target coordinate budget of 4.5 bits. Pre, Post, and Joint denote TORQUE retention before rotation, after rotation, and at both stages, respectively.

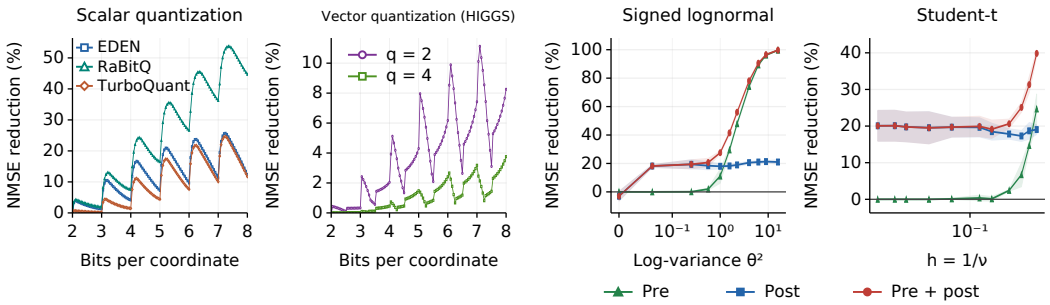
3.3 Encoding and Decoding Time

Figure 4 shows encoding and decoding times for normalized Gaussian and Student-t_3 inputs on an NVIDIA RTX 5090. Encoding includes optimization. Our C++17/CUDA implementation uses randomized Hadamard rotations and fused kernels that keep intermediate values in shared memory.

We compile with NVCC 13.3 for `sm_120`, MSVC 19.44, and `-O3 --use_fast_math`. Launch configurations are tuned separately for each variant on a held-out seed and fixed before measurement. We average five seeds and report 95% confidence intervals.

At $d = 256$, batch size 128, and a 4.5-bit coordinate budget, encoding time relative to the corresponding baseline increases by 48.9–73.7% with pre-rotation retention and by 48.4–99.7% with joint retention, across the three quantizers and both input distributions. For post-rotation retention, the encoding-time increase is modest and always under 15.7%. Across all three retention variants, decoding time is barely affected.

Additional results for native CPU and Apple GPU implementations appear in Appendix D.



(a) Gaussian-model: scalar and vector quantization.

(b) Improvement breakdown compared to EDEN.

Figure 5: NMSE reductions from applying TORQUE.

3.4 Gaussian-model error and retention choices

Figure 5(a) isolates the benefit of retention after rotation under the Gaussian model. Since the post-rotation distribution is independent of the input, this corresponds to the *worst-case* attainable NMSE, up to an additive factor that decays with the dimension Vargaftik et al. (2022). We compare each quantizer with and without retention. The scalar curves use EDEN, RaBitQ, and TurboQuant on standard Gaussian coordinates. The vector curves use HIGGS on blocks $G \sim N(0, I_q)$ of $q = 2$ or $q = 4$ coordinates. HIGGS maps each block to its nearest codeword in Euclidean distance. For each c, s, q combination, we fit a codebook offline for the induced truncated Gaussian distribution.

At each budget, we select the threshold and bit allocation with the smallest estimated error among the feasible candidates, including no retention ($c = \infty$).

At the same expected bit budget, TORQUE reduces scalar NMSE by up to 25.69% for EDEN, 53.73% for RaBitQ, and 24.65% for TurboQuant. For HIGGS, the plotted NMSE reductions reach 11.12% for $q = 2$ and 3.76% for $q = 4$. Again, we see the benefit of using slightly-more-than-integral values for b, which allows TORQUE to use the fractional leftover for the outliers.

Figure 5(b) examines the benefit of retaining large input coordinates before rotation based on the skewness of the input distribution. We compare EDEN with retention before, after, or at both stages on signed-lognormal and Student-t inputs. Signed-lognormal coordinates have independent random signs and log-magnitudes drawn from $N(0, \theta^2)$. Student-t coordinates have ν degrees of freedom, plotted as $h = 1/\nu$. Increasing θ^2 or h makes large coordinates more pronounced. For each of five seeds, we draw 256 vectors of dimension 256 and normalize them to unit norm. All retention choices share the same inputs and randomized Hadamard rotations. The budget is eight bits per coordinate.

Relative to EDEN, TORQUE reduces NMSE by up 99.68%. The improvement's source depends on input skewness: when the data is heavy-tailed (low θ^2 or h), TORQUE's improvement stems from post-rotation retention. In contrast, when the data is skewed, the pre-rotation retention does most of the heavy lifting. Together, these results illustrate how the two stages can complement each other.

4 Limitations and conclusion

We presented TORQUE, a framework that jointly optimizes outlier retention before and after rotation under a shared bit budget. Our evaluation shows that TORQUE improves the reconstruction accuracy of state-of-the-art scalar and vector quantizers across different input distributions and applications.

The main computational cost is the additional encoding time required for online optimization. This relative overhead decreases as the vector dimension grows, while decoding time remains close to that of the underlying quantizer. The encoding cost is less important in applications where data is quantized once and decoded many times, making decoding latency the main concern. Examples include KV caches for reusable system prompts and RAG contexts, as well as post-training weight quantization.

An interesting direction for future work is to to reduce the optimization cost by exploiting slowly changing input distributions (e.g., gradients over different training rounds) or sampling a small number of chunks out of a large vector we wish to quantize for optimizing the parameters.

AI USE STATEMENT

Generative AI **was used** to help establish and write the theoretical result in Appendix A, edit the paper for readability, and implement and evaluate methods.

Generative AI **was not used** to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, or translate.

We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 3 documents the publicly available datasets and parameters of the synthetic distributions, preprocessing, evaluation metrics, bit accounting, and procedures for estimating uncertainty used in our experiments. The experimental appendices provide additional evaluation protocols and results. Appendix A presents the assumptions and complete proof of the top-k retention result, while Appendix B describes alternative implementations and their design choices. These descriptions support verification of our theoretical claims and reproduction of the reported experimental comparisons.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 100213–100240, 2024. doi: 10.52202/079017-3180. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/b5b939436789f76f08b9d0da5e81af7c-Abstract-Conference.html.

Ran Ben-Basat, Shay Vargaftik, Amit Portnoy, Gil Einziger, Yaniv Ben-Itzhak, and Michael Mitzenmacher. Accelerating federated learning with quick distributed mean estimation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 3410–3442. PMLR, 2024. URL https://proceedings.mlr.press/v235/ben-basat24a.html.

Ran Ben-Basat, William Kuszmaul, Michael Mitzenmacher, Amit Portnoy, and Shay Vargaftik. Quantizing with randomized Hadamard transforms: Efficient heuristic now proven. In *Advances in Neural Information Processing Systems*, 2026. URL https://arxiv.org/abs/2605.06014. Accepted for publication.

bitsandbytes contributors. bitsandbytes: LLM.int8(). Software documentation, 2026. URL https://huggingface.co/docs/bitsandbytes/reference/nn/linear8bit. Accessed September 26, 2026.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-bit quantization of large language models with guarantees. In *Advances in Neural Information Processing Systems*, volume 36, pp. 4396–4429, 2023. doi: 10.52202/075280-0196. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/0df38cd13520747e1e64e5b123a78ef8-Abstract-Conference.html.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30318–30332, 2022. doi: 10.52202/068431-2198. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c3ba4962c05c49636d4c6206a97e9c8a-Paper-Conference.pdf.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. SpQR: A sparse-quantized representation for near-lossless LLM weight compression. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/

2024/hash/1787533e171dcc8549cc2eb5a4840eec-Abstract-Conference.html.

Ying Feng, Piotr Indyk, Michael Kapralov, Dmitry Krachun, and Boris Prokhorov. Provable quantization with randomized Hadamard transform. *arXiv preprint arXiv:2605.13810*, 2026. URL https://arxiv.org/abs/2605.13810.

Jianyang Gao and Cheng Long. RaBitQ: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 2(3):167:1–167:27, 2024. doi: 10.1145/3654970.

Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. Practical and asymptotically optimal quantization of high-dimensional vectors in Euclidean space for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 3(3): 1–26, 2025. doi: 10.1145/3725413.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. OWQ: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(12):13355–13364, 2024. doi: 10.1609/aaai.v38i12.29237. URL https://ojs.aaai.org/index.php/AAAI/article/view/29237.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 87–100, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. SpinQuant: LLM quantization with learned rotations. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/e5b1c0d4866f72393c522c8a00eed4eb-Abstract-Conference.html.

LLM Compressor. LLM Compressor: Applying transforms to improve quantization accuracy. Software documentation, 2026. URL https://github.com/vllm-project/llm-compressor/blob/main/examples/transform/README.md. Accessed September 26, 2026.

Vladimir Malinovskii, Andrei Panferov, Ivan Ilin, Han Guo, Peter Richtárik, and Dan Alistarh. HIGGS: Pushing the limits of large language model quantization via the linearity theorem. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 10857–10886. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.543. URL https://aclanthology.org/2025.naacl-long.543/.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Andrei Panferov, Erik Schultheis, Soroush Tabesh, and Dan Alistarh. Quartet II: Accurate LLM pre-training in NVFP4 by improved unbiased gradient estimation. *arXiv preprint arXiv:2601.22813*, 2026. URL https://arxiv.org/abs/2601.22813.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Ananda Theertha Suresh, Felix X. Yu, Sanjiv Kumar, and H. Brendan McMahan. Distributed mean estimation with limited communication. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3329–3337. PMLR, 2017. URL https://proceedings.mlr.press/v70/suresh17a.html.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even better LLM quantization with Hadamard incoherence and lattice codebooks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 48630–48656. PMLR, 2024. URL https://proceedings.mlr.press/v235/tseng24a.html.

Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben-Itzhak, and Michael Mitzenmacher. DRIVE: One-bit distributed mean estimation. In *Advances in Neural Information Processing Systems*, volume 34, pp. 362–377, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/0397758f8990c1b41b81b43ac389ab9f-Abstract.html.

Shay Vargaftik, Ran Ben Basat, Amit Portnoy, Gal Mendelson, Yaniv Ben Itzhak, and Michael Mitzenmacher. EDEN: Communication-efficient and robust distributed mean estimation for federated learning. In *International Conference on Machine Learning*, pp. 21984–22014. PMLR, 2022.

vLLM. TurboQuant: KV-cache quantization for vLLM. Software documentation, 2026. URL https://docs.vllm.ai/en/latest/api/vllm/model_executor/layers/quantization/turboquant/. Accessed September 26, 2026.

Yongyi Yang, Jianyang Gao, and Wei Hu. RaanA: A Fast, Flexible, and Data-Efficient Post-Training Quantization Algorithm. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=IsgrZr2j1R.

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. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/5c802ef38ab6e366c2ea06eee554c088-Abstract-Conference.html.

Tomer Zilca and Gal Mendelson. Approximating uniform random rotations by two-block structured Hadamard rotations in high dimensions. *arXiv preprint arXiv:2604.23418*, 2026. URL https://arxiv.org/abs/2604.23418.

A Optimal retention before rotation

We justify the top-k rule used in Section 2. The argument uses the rotation and exact-retention assumptions stated there. We first fix the number of retained coordinates and compare the possible sets of positions. Each set must have the same storage cost and allow the same quantizer parameters.

Error for a chosen set of positions. Let $x \neq 0$ be the input, let $\mathcal{I} \subseteq \{1, \dots, d\}$ contain k retained positions, and let $h_{\mathcal{I}}$ contain the input values at those positions and zeros elsewhere. The fraction of the squared input norm left in the remaining vector is

$$\rho_{\mathcal{I}} = \frac{\|x - h_{\mathcal{I}}\|_2^2}{\|x\|_2^2} = 1 - \frac{\sum_{i \in \mathcal{I}} x_i^2}{\|x\|_2^2}.$$

When $\rho_{\mathcal{I}} > 0$, we normalize the remaining vector as

$$z_{\mathcal{I}} = \frac{\sqrt{d}}{\|x - h_{\mathcal{I}}\|_2}(x - h_{\mathcal{I}}), \qquad \|z_{\mathcal{I}}\|_2^2 = d.$$

For fixed parameters (c, s), let $\widehat{z}_{\mathcal{I}}$ be its reconstruction. The rotation makes its expected normalized squared error independent of the direction of $z_{\mathcal{I}}$, so

$$\frac{1}{d}\mathbb{E}\|\widehat{z}_{\mathcal{I}} - z_{\mathcal{I}}\|_2^2 = \epsilon_d(c, s).$$

Restoring the scale and adding the retained values gives the intermediate reconstruction

$$\widetilde{x} = h_{\mathcal{I}} + \frac{\|x - h_{\mathcal{I}}\|_2}{\sqrt{d}}\widehat{z}_{\mathcal{I}}.$$

The input has the same expression with $z_{\mathcal{I}}$ in place of $\widehat{z}_{\mathcal{I}}$, so the exactly retained part cancels when we subtract them. The decoder then restores the retained positions to their exact input values, giving $\widehat{x}$. This replaces the error at those positions by zero and can only decrease the squared error. Therefore,

$$\text{NMSE} = \frac{\mathbb{E}\|\widehat{x} - x\|_2^2}{\|x\|_2^2} \leq \frac{\mathbb{E}\|\widetilde{x} - x\|_2^2}{\|x\|_2^2} = \rho_{\mathcal{I}}\, \epsilon_d(c, s). \tag{6}$$

If $\rho_{\mathcal{I}} = 0$, the retained values already specify x and the error is zero. If any set of k positions has this property, the top-k set also does and achieves zero error at the same retention cost. The remaining argument concerns nonzero remaining vectors.

Choosing the retained positions. For fixed k, retention costs kL_{pre} bits. The dimension, post-rotation position cost, and remaining budget are unchanged when we replace one set of retained positions with another. Thus, the feasible pairs (c, s) are the same. For each pair, only $\rho_{\mathcal{I}}$ in the bound in Equation (6) depends on the chosen positions.

Proposition 1 (Optimal retention for the error bound)**.** *Assume exact retention and a nonnegative, direction-independent expected normalized squared error $\epsilon_d(c, s)$ for the normalized remaining vector. Suppose all sets of k retained positions have the same storage cost and feasible quantizer parameters. Then retaining the k largest-magnitude input coordinates minimizes the bound*

$$\left(1 - \frac{\sum_{i\in\mathcal{I}} x_i^2}{\|x\|_2^2}\right)\epsilon_d(c, s)$$

over all sets $\mathcal{I}$ of size k, for every feasible pair (c, s). Consequently, the search over k, c, s in Equation (5) *needs no additional search over position sets.*

Proof. Suppose $i \in \mathcal{I}$ and $j \notin \mathcal{I}$ satisfy $x_i^2 < x_j^2$. Replacing i with j reduces $\rho_{\mathcal{I}}$ by $(x_j^2 - x_i^2)/\|x\|_2^2$. Since $\epsilon_d(c, s) \geq 0$, this replacement cannot increase the bound. It also leaves the storage cost and feasible parameters unchanged.

Repeating these replacements gives the k largest squared coordinates. This set minimizes the bound for every feasible pair (c, s), and therefore also after selecting the best such pair. □

Since $k = 0$ is included in the joint search, the minimum objective therefore cannot exceed that of the best feasible candidate using retention only after rotation. The search may select $k = 0$ when retaining input coordinates offers no improvement.

Other dimensions and error estimates. The same argument applies when retained coordinates are removed before rotation. For fixed k, the dimension $m = d - k$ is the same for every position set. We normalize the remaining vector to squared norm m and replace $\epsilon_d(c, s)$ by $\epsilon_m(c, s)$. The same top-k conclusion follows. For vector quantization, this option requires q to divide m; when no nonzero coordinates remain, rotation and quantization are skipped.

B Alternative implementation choices

The formulation in Section 2 keeps all d coordinates and uses fixed-width indices. Every choice of k can therefore use the same rotation implementation and precomputed tables of errors and expected bit costs. The retained positions are also simple to store and recover. Together with truncated Gaussian codebooks constructed offline, these choices offer a useful compromise between storage efficiency and a fast practical implementation. Below, we discuss other implementation options, not used in our evaluation, that may offer a better accuracy to representation size trade-off at the cost of increased complexity and storage.

Dimension and reconstruction. One option is to remove the retained input coordinates and collect the remaining coordinates into a smaller vector. Write m for the dimension entering the rotation: the main formulation uses $m = d$, while this option gives $m = d - k$. For a nonzero remaining vector, we normalize the smaller vector to obtain $z_k \in \mathbb{R}^m$ with $\|z_k\|_2^2 = m$, and write $U = Rz_k$. The decoder rescales by $\|x_{-k}\|_2/\sqrt{m}$ and reinserts the retained input values at their original positions. The expected bit cost is then

$$\mathcal{B}(k, c, s) = kL_{\text{pre}} + m\big[p_m(c)L_{\text{post}} + (1 - p_m(c))s\big]$$

and the objective becomes $\rho_k \epsilon_m(c, s)$, where $p_m(c)$ and $\epsilon_m(c, s)$ are the expected retained fraction and NMSE at dimension m. We substitute them into Equation (5). For vector quantization, replace $p_m(c) = p_{m,1}(c)$ by $p_{m,q}(c) = \mathbb{P}(\|U_j\|_2 > c)$ and consider only choices for which q divides m. This option processes $d - k$ coordinates, but the rotation size and finite-dimensional tables now depend on k.

Since $\epsilon_m(c, s)$ depends on k when $m = d-k$, the largest feasible count need not minimize $\rho_k \epsilon_m(c, s)$. The one-count-per-pair reduction in Section 2.3 therefore does not automatically apply. Enumerating the admissible counts with precomputed cost and error tables takes $O(d \log d + d|C||S|)$ time.

When keeping $m = d$, the decoder sets the retained positions of $\widehat{z}_k$ to zero before adding the retained values, as described in Section 2.3. This removes their reconstruction noise, so $\rho_k \epsilon_d(c, s)$ is an upper bound on the final NMSE.

Value precision and index widths. For scalar retention with fixed-width indices, the position costs are $p_{\text{pre}} = \lceil \log_2 d \rceil$ and $p_{\text{post}} = \lceil \log_2 m \rceil$. With $m = d$, a simple choice is to use the same precision and index width at both stages: $v = v_{\text{pre}} = v_{\text{post}}$ and $p = p_{\text{pre}} = p_{\text{post}} = \lceil \log_2 d \rceil$. The two stages may also use different value precisions, with their costs included in L_{pre} and L_{post}. For blocks of q coordinates, one index identifies a whole block. The post-rotation cost per retained coordinate is then $v_{\text{post}} + \lceil \log_2(m/q) \rceil / q$.

Position encodings. When few input coordinates are retained, we can store differences between consecutive positions or encode the whole subset. These choices can use $p \approx \log_2(d/k)$ bits per position on average, plus overhead.

We can also store retained values in their original order and record where to reinsert them among the $d - k$ remaining entries. Each gap index uses $\lceil \log_2(d - k + 1) \rceil$ bits, approximately $\log_2(d - k)$ when $d - k$ is large.

These choices can save index bits but require more coding work. The top-k result in Appendix A requires equal-size position sets to have the same storage cost and feasible quantizer parameters. At fixed dimension and fixed (c, s), a cost that depends only on k preserves the largest-feasible-count rule, but requires solving feasibility for that cost instead of using the fixed-width formula. If the cost depends on the retained positions themselves, the top-k guarantee need not hold.

Encoding block positions jointly. Suppose r of the m/q blocks are retained. There are $\binom{m/q}{r}$ possible sets of retained positions. Enumerative coding stores the index of the r-sized subset, which requires $\lceil \log_2 \binom{m/q}{r} \rceil$ bits, the minimum fixed-length cost when r is known. The decoder must also know r, which can be inferred from the representation or stored using $O(\log m)$ additional bits.

Under the Gaussian model (§2.2), a block is retained with probability $\pi = \mathbb{P}(\|G\|_2 > c)$. Joint encoding uses roughly $H(\pi)$ bits per block for its position, or $H(\pi)/q$ bits per coordinate. Retained coordinates use v bits each, while inlier coordinates use s bits each. Adding these costs gives

$$b_G(c, s) = \frac{H(\pi)}{q} + v\pi + (1 - \pi)s, \qquad H(\pi) = -\pi \log_2 \pi - (1 - \pi) \log_2(1 - \pi). \tag{7}$$

Here, H is the binary entropy function, with $H(0) = H(1) = 0$. The three terms account for positions, retained values, and inlier codes.

For fixed q, c, the expected cost per coordinate is at most $b_G + O(\log m/m)$. This bound does not require independent blocks. Including retention before rotation, the expected total cost is therefore at most $kL_{\text{pre}} + mb_G(c, s) + O(\log m)$.

Dimension-dependent codebooks. We use codebooks optimized for a truncated Gaussian distribution, making their design independent of the input dimension.

An alternative is to optimize codebooks for the exact distribution induced by the rotation: a scaled, shifted Beta distribution for scalar quantization, or the joint spherical marginal for vector quantization. For each threshold c, we condition this distribution on the coordinates or blocks that are not retained, then design one codebook for each bit allocation s. At a fixed rotation dimension d, this gives $|C||S|$ codebooks, shared by all pre-rotation choices of k. If we remove the retained input coordinates, the dimension becomes $d - k$. We then repeat this construction for each supported nonzero dimension, giving up to $d|C||S|$ codebooks.

Another option is to retain a fixed number of the largest-magnitude coordinates after rotation instead of using a threshold. This fixes the retention count and, with fixed-width encoding, makes the bit cost deterministic. However, the inlier distribution now depends on both the rotation dimension and the retention count. We therefore design one codebook for each supported retention count and bit allocation at each dimension. Supporting all counts gives up to $d|S|$ codebooks at a fixed dimension, or $O(d^2|S|)$ if the dimension also varies as $d - k$.

C Additional application results

We extend the application experiments in Section 3 to biased reconstruction and GIST retrieval.

C.1 Biased Activation and KV-cache compression

Figure 6 complements Figure 2 with EDEN-B, RaBitQ-B, and TurboQuant-MSE, the biased variants of these quantizers. We evaluate the activations using the setup in Section 3.1. For KV caches, we use 256 contexts and retention only after rotation.

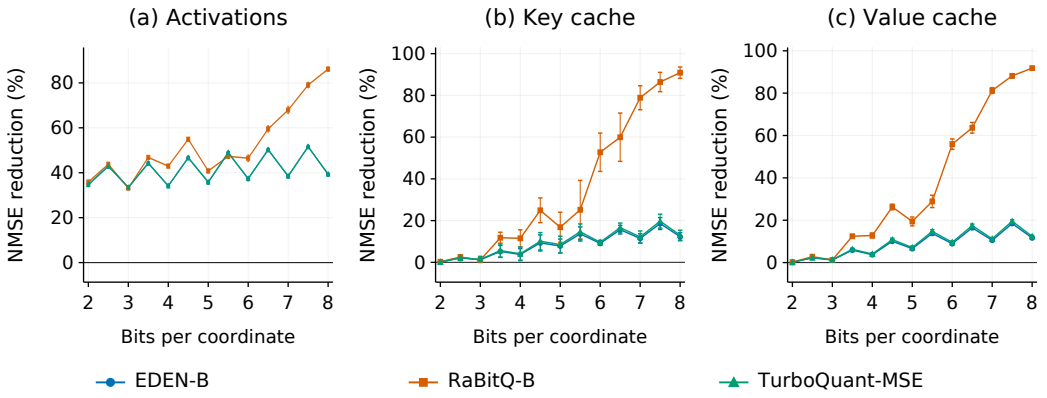


Figure 6: Biased reconstruction: NMSE reductions for activations, key caches, and value caches. The KV-cache results use retention only after rotation.

C.2 Additional retrieval results

GIST retrieval. We extend the scalar comparison in Section 3.2 to squared Euclidean search on GIST-960 (Jegou et al., 2010), using its complete index of one million vectors and all 1,000 queries.

For unbiased GIST reconstruction, we store the original squared norm and include its cost in the metadata. For a query y and an index vector x, the distance estimate is

$$\|y\|_2^2 + \|x\|_2^2 - 2\langle y, \widehat{x}\rangle.$$

It uses the stored $\|x\|_2^2$, rather than the squared norm of the reconstruction.

Figure 7 is the GIST counterpart of Figure 3. It shows the 13 half-bit budgets; the full sweep also includes the intervening quarter-bit measurements. Lower reconstruction NMSE does not consistently improve Recall@1.

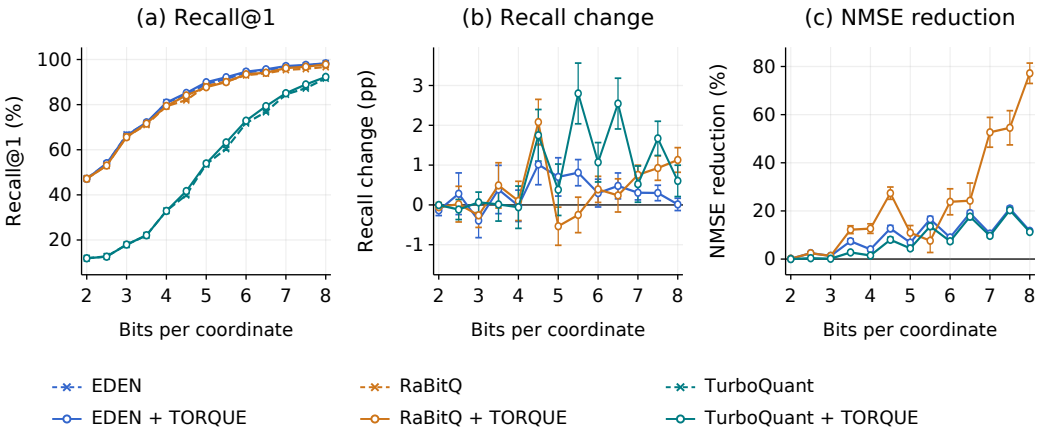


Figure 7: GIST retrieval: Recall@1, recall change, and NMSE reduction.

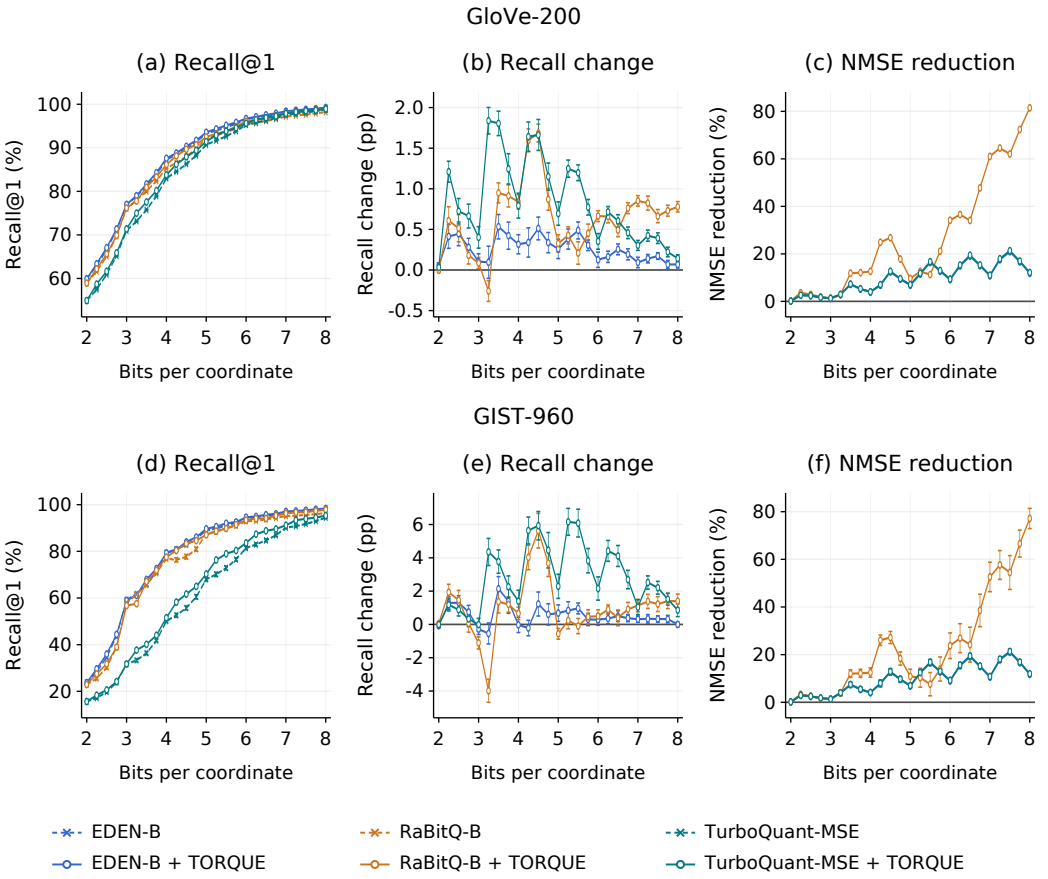


Figure 8: Biased retrieval on GloVe and GIST: Recall@1, recall change, and NMSE reduction.

Biased reconstructions. Figure 8 evaluates EDEN-B, RaBitQ-B, and TurboQuant-MSE on GloVe and GIST. We use the same seeds, budgets, and retention stage as the unbiased comparisons. Combining the three biased and three unbiased variants gives $4{,}650$ comparisons per dataset over 775 budget–seed configurations.

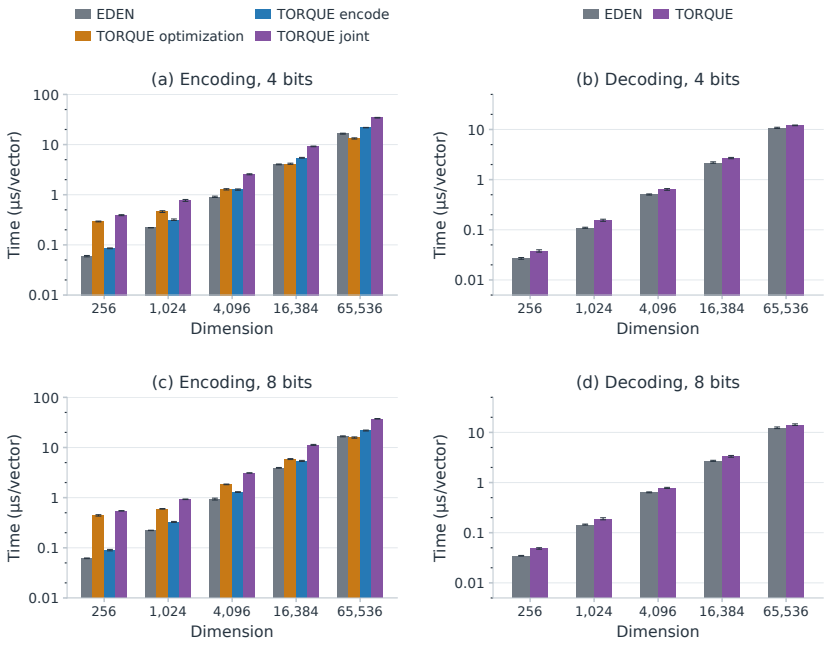


Figure 9: Native CPU timing: Encoding and decoding at four bits and eight bits.

D NATIVE CPU AND APPLE GPU TIMING

We extend Section 3.3 with runtime evaluation of TORQUE for CPU and Apple GPU. All TORQUE measurements here use both retention stages. Figures 9 and 10 separate parameter selection (TORQUE *optimization*), encoding with selected parameters (TORQUE *encode*), and their directly timed combination (TORQUE *joint*). The decoding plots compare complete reconstruction times.

Hardware and implementation. We use an Apple M2 Max with eight performance cores, four efficiency cores, a 30-core GPU, and 32 GiB of memory, running macOS 15.7.2. The CPU implementation uses C++17, NEON instructions, and twelve persistent workers; Apple Clang 15.0.0 compiles it with `-O3 -mcpu=native -ffast-math`. The GPU uses Metal kernels with fast math enabled. Inputs, codebooks, parameter selection, rotations, and reconstruction use FP16 on both platforms, including norm and correction sums. Power-of-two rescaling limits overflow and underflow. The CPU keeps conservative rounding for its selection bounds. Rotations use cache blocks on the CPU and shared-memory tiles on the GPU; indices and bit packing use integer arithmetic.

Reducing the sorting work. The GPU, and the CPU for $d \geq 1024$, group coordinates into 256 buckets by magnitude. The buckets describe how large the coordinates are without sorting them individually. For a proposed retained count k, some buckets are retained completely, some remain completely, and at most one is split. Each bucket's range bounds the sum of squares of its remaining coordinates. For example, ten coordinates with magnitudes between 0.10 and 0.12 contribute between 0.10 and 0.144. Combining these bounds gives an interval for the candidate's objective $\rho_k \epsilon_d(c, s)$. If one candidate's lower bound exceeds another's upper bound, it cannot win and can be skipped. The remaining choices often require only a small subset of coordinates to be sorted. When the bounds overlap, we keep both candidates and compare them using the actual values.

Inputs. We test $d \in \{256, 1024, 4096, 16384, 65536\}$, with $2^{23}/d$ vectors per batch. Before timing, we generate independent Student-t_3 coordinates, normalize each vector in FP64, and store it in FP32. Both implementations convert the inputs to FP16 before timing. Seeds are $17, 29, 43, 71, 101$. NumPy 2.5.3 uses PCG64 with `SeedSequence([seed,d,0])` for inputs

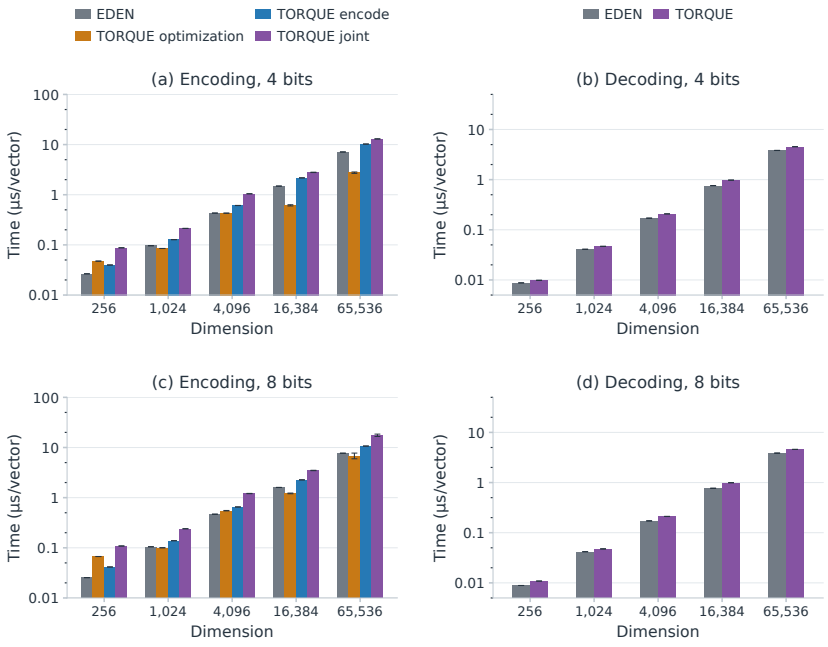


Figure 10: Apple GPU timing, with the same layout as Figure 9.

and `SeedSequence([seed,d,1])` for rotation signs. Both methods receive the same inputs and randomized Hadamard rotations.

Budgets and codebooks. We use budgets of $b = 4$ and $b = 8$ expected bits per coordinate for retained values, their indices, and quantizer codes. For these measurements, both EDEN and TORQUE reserve an additional 160 bits per vector for metadata, excluded from b. This stores norms, a reconstruction correction factor, retained counts, and a quantizer identifier, and includes padding.

Retained values use FP16. Indices use eight bits at $d = 256$ and sixteen bits at larger dimensions. Thus, for $d > 256$, each retained value and its index fit in one 32-bit word.

We compute the codebooks offline using

$$S = \{1 + j/64 : j = 0, \ldots, 448\},$$
$$C = \{1.75, 2, 2.25, 2.5, 2.75, 3, 3.5, 4, 4.5, 5, 6, \infty\}.$$

For each dimension and budget, we also precompute a list of feasible parameter choices (k, c, s) and their error estimates. At runtime, we use the input to compare these choices and select the parameters.

Measurements. We measure the time to process a batch and divide by its number of vectors. After warm-up, we take the median runtime for each seed and average across five seeds. Error bars show 95% bootstrap confidence intervals.

CPU timings include synchronization between workers. GPU timings cover execution on the device, with data already in GPU memory. Each GPU timing sample processes 16 batches and reports the average time per batch. Setup, host submission, and data transfers are excluded.

Results. Across the tested dimensions and budgets, the additional runtime cost is concentrated in encoding, which includes online parameter selection. With the parameters already selected, TORQUE encoding takes 1.30–1.48× EDEN's encoding time on the CPU and 1.32–1.63× on the GPU. Including online selection increases these ranges to 2.08–8.85× and 1.82–4.28×, respectively. The relative overhead is largest at $d = 256$ and lower at every larger dimension tested. Decoding remains closer to EDEN throughout, taking 1.12–1.41× its decoding time on the CPU and 1.13–1.29× on the GPU.