DalQ: Reconciling Accuracy and Efficiency in Vector Quantization

Hao Zheng[1,2], Xiaodong Lee[1,3,4] (✉), Botao Peng[1], Tianqi Hou[5], and Themis Palpanas[6]

[1] Institute of Computing Technology, Chinese Academy of Sciences, China
{zhenghao21b,xl,pengbotao}@ict.ac.cn
[2] University of Chinese Academy of Sciences, China
[3] Fuxi Institution, China
[4] Center for Internet Governance, Tsinghua University, China
[5] Huawei Technologies Co., Ltd., China houtianqi2@huawei.com
[6] Université Paris Cité, France themis@mi.parisdescartes.fr

Abstract. Vector quantization is a fundamental technique for reducing memory footprint and computational overhead in modern database systems, search engines, and large language models. However, existing vector quantization methods remain limited in simultaneously achieving high quantization accuracy and computational efficiency. In this paper, we propose Distribution-aware locally-adaptive Quantization (DalQ), which achieves superior accuracy with competitive efficiency through two key mechanisms. First, distribution-aware range adaptation leverages Gaussian-like embedding distributions via an adaptive clip factor, concentrating precision in high-density regions while tolerating controlled clipping of rare outliers. Second, decomposed vector refinement employs a two-stage geometric optimization strategy for near-optimal reconstruction: directional alignment via iterative code search and magnitude refinement via projection-based scaling. We prove the existence and uniqueness of the optimal clip factor and identify a reference clip factor enabling tuning-free deployment. Extensive evaluations demonstrate the substantial advantages of DalQ over state-of-the-art baselines. Against LVQ, DalQ achieves up to 96.6% reduction in quantization error with competitive efficiency. Against RaBitQ, DalQ not only reduces quantization error by up to 63.4%, but also delivers up to 25.8× faster quantization and up to 3.2× faster distance computation.

Keywords: Vector Quantization · Approximate Nearest Neighbor Search

1 Introduction

Vector quantization is a fundamental compression technique that transforms continuous vectors into compact discrete representations, reducing memory and computational overhead across modern database systems [22,14], search engines [26,21], and large language models [12,5,18]. The explosive growth of high-dimensional embeddings [12] has made efficient compression increasingly critical—

from billion-scale similarity search [1] to deploying models with hundreds of billions of parameters on resource-constrained hardware [9]. The core challenge is universal: how to compress high-dimensional vectors with minimal information loss while maintaining computational efficiency.

Existing vector quantization methods fall into two families. Product Quantization (PQ) methods [17,13,3,20] require costly codebook training and slow lookup-table distance computation. Scalar Quantization (SQ) methods avoid codebook training by quantizing each dimension independently, yet no existing approach achieves both high accuracy and efficiency: classical SQ [8] underutilizes quantization ranges as vectors occupy only small fractions of global bounds; LVQ [1] mitigates this with per-vector bounds but ignores value distributions; RaBitQ [11,10] improves accuracy via an unbiased estimator but incurs high overhead and supports only limited distance metrics.

In this paper, we propose DalQ (Distribution-aware locally-adaptive Quantization), motivated by the observation that dimension values in deep learning embeddings typically follow approximately Gaussian distributions with density concentrated in central regions; for irregular distributions, random orthogonal rotation [6] serves as an effective distribution regularizer. DalQ introduces two mechanisms. First, distribution-aware range adaptation introduces a clip factor $\beta \in (0, 1]$ that adaptively tightens quantization ranges from full per-vector bounds to narrower clipped ranges, trading controlled clipping error in low-density outliers for reduced quantization error in the high-density majority. We prove the existence and uniqueness of the optimal clip factor, develop an efficient adaptive search algorithm, and identify a reference factor providing tuning-free deployment. Second, decomposed vector refinement employs a two-stage geometric optimization strategy beyond standard value-proximity rounding: (1) directional alignment via iterative code search maximizes cosine similarity through greedy code search; (2) magnitude refinement via projection-based scaling computes the optimal scalar for magnitude refinement, achieving near-optimal reconstruction within discrete constraints. We summarize our contributions as follows:

- We empirically observe that modern embeddings exhibit Gaussian-like distributions with values concentrated in central regions, and that random orthogonal rotation serves as an effective regularizer, transforming arbitrary distributions toward Gaussian forms.
- We propose DalQ, a vector quantization method achieving high accuracy via two mechanisms: (1) *distribution-aware range adaptation* employs an adaptive clip factor to concentrate quantization precision in high-density regions while tolerating controlled clipping of rare outliers; (2) *decomposed vector refinement* applies a two-stage geometric optimization strategy for near-optimal reconstruction: directional alignment and magnitude refinement.
- We prove the existence and uniqueness of the optimal clip factor under a unimodal distribution assumption, develop an adaptive search algorithm to locate a near-optimal clip factor, and identify a reference factor $\beta_{\text{ref}} = 1-e^{-b}$ enabling tuning-free deployment or effective search initialization.

– We conduct extensive evaluations on large-scale datasets spanning diverse domains and dimensionalities. Compared to LVQ, DalQ achieves up to 96.6% lower MSE while maintaining competitive efficiency. Compared to RaBitQ, DalQ achieves up to 63.4% lower MSE, up to 25.8× faster quantization, and up to 3.2× faster distance computation.

2 Preliminaries

Scalar quantization. Scalar quantization compresses each dimension of a vector independently by mapping floating-point values to discrete integer codes. Given a vector $\mathbf{z} \in \mathbb{R}^d$, uniform b-bit scalar quantization encodes each dimension as an integer code $c_j \in \{0, 1, \ldots, 2^b - 1\}$:

$$c_j = Q(z_j, l, u, b) = \left\lfloor \frac{z_j - l}{u - l} \cdot (2^b - 1) + 2^{-1} \right\rfloor \tag{1}$$

The reconstruction value is computed via dequantization:

$$\hat{z}_j = Q^{-1}(c_j, l, u, b) = l + c_j \cdot \Delta, \qquad \Delta = \frac{u - l}{2^b - 1} \tag{2}$$

where l and u are the lower and upper bounds calculated based on predefined criteria, and Δ is the step size. The per-dimension reconstruction error is bounded by $|z_j - \hat{z}_j| \leq \Delta/2$.

3 Method: DalQ

We now present the method DalQ (Distribution-aware locally-adaptive Quantization). We begin by presenting distribution-aware range adaptation, then describe decomposed vector refinement, and finally provide the complete quantization procedure with detailed algorithms.

3.1 Distribution-Aware Range Adaptation

Distribution Observation. The design of DalQ is motivated by a fundamental distributional property of modern deep learning embeddings: after mean-centering, dimension values exhibit Gaussian-like marginal distributions with density concentrated near the center and decaying rapidly toward the tails, as illustrated in Figure 1 across four representative datasets. This pattern is theoretically grounded: repeated linear transformations and non-linear activations in neural networks drive dimension values toward Gaussian by the Central Limit Theorem [4], a tendency further reinforced by common normalization techniques such as batch normalization [16].

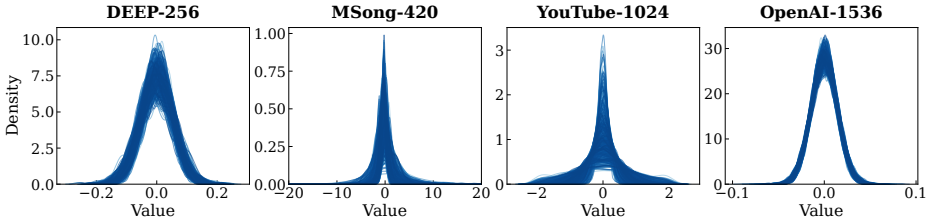


Fig. 1: Kernel density estimates of 1000 randomly sampled embedding vectors from four datasets. Each curve represents the marginal distribution across all dimensions of one vector.

Random Rotation for Distribution Regularization. Real-world embeddings sometimes exhibit irregular distributional patterns (e.g., asymmetry, heavy tails). To address this, we propose applying random orthogonal rotation $\mathbf{z}' = \mathbf{R}\mathbf{z}$ as a preprocessing regularizer, where $\mathbf{R}$ is obtained via QR decomposition of a matrix with i.i.d. $\mathcal{N}(0, 1)$ entries. As shown in Figure 2, random rotation transforms a skewed per-vector distribution into an approximately symmetric, zero-centered one. This effect arises because each rotated dimension $z'_i = \sum_j R_{ij} z_j$ is a weighted sum of all original dimensions with $\sum_j R_{ij}^2 = 1$, whose marginal distribution converges toward Gaussian forms by the Central Limit Theorem [4] as dimensionality d increases. Therefore, random rotation can serve as an optional preprocessing step for datasets with irregular distributions.

Implications for Quantization. The reconstruction error of a quantization method is given by:

$$\text{MSE} = \frac{1}{d}\sum_{j=1}^{d}(z_j - \hat{z}_j)^2 = \int_{-\infty}^{+\infty}(z - \hat{z}(z))^2 p(z)\,dz \tag{3}$$

where $\hat{z}$ denotes the reconstructed value for original z. Different regions of the value range contribute unequally to the overall reconstruction error, weighted by

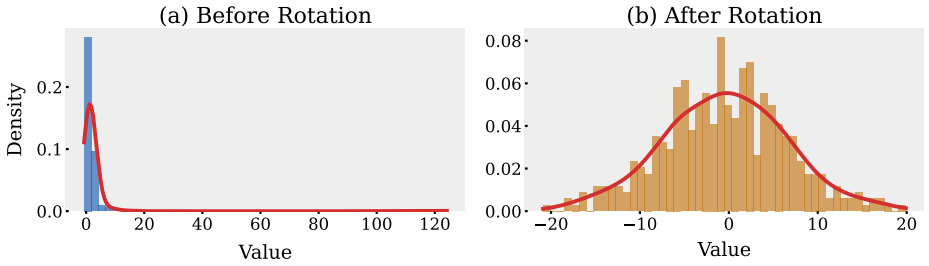
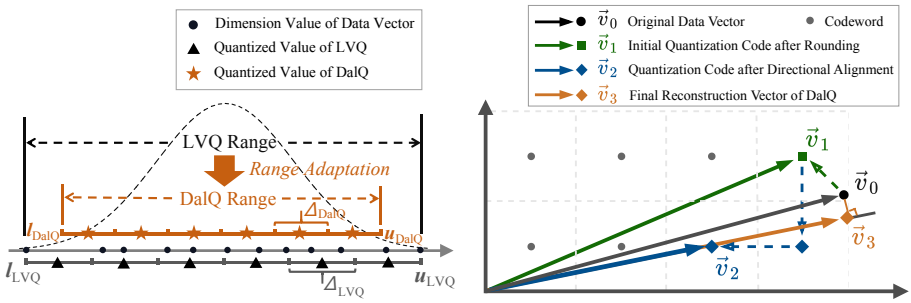


Fig. 2: Marginal distribution of a randomly sampled vector from the MSong dataset before (left) and after (right) random orthogonal rotation.



(a) Range Adaptation (b) Decomposed Vector Refinement

Fig. 3: Illustration of the two key mechanisms of DalQ. (a) Range Adaptation tightens both the quantization range and the step size. (b) Decomposed Vector Refinement reduces the quantization Mean Squared Error (MSE), as evidenced by $\|v_0 - v_3\|_2 < \|v_0 - v_1\|_2$.

their probability density $p(z)$. Under Gaussian-like distributions, values in the central region are far more probable than those in the periphery. Consequently, allocating finer quantization to the center—even at the cost of clipping peripheral outliers—can substantially reduce the total reconstruction error.

Range Adaptation. DalQ exploits this distributional characteristic by introducing the *clip factor* $\beta \in (0, 1]$, which adaptively tightens the quantization range from the full per-vector bounds $[l, u]$ to a narrower clipped range centered within the original bounds (Figure 3a).

Definition 1 (Clip Factor). *For a vector* $\mathbf{z} \in \mathbb{R}^d$ *with original per-vector bounds* $l = \min_{j\in[d]} z_j$ *and* $u = \max_{j\in[d]} z_j$*, the clip factor* $\beta \in (0, 1]$ *determines the clipped bounds* $[l^*, u^*]$ *according to:* $l^* = l + \frac{1-\beta}{2}W, u^* = u - \frac{1-\beta}{2}W$*, where* $W = u - l$ *is the original range width. The effective quantization range width after clipping is* $W^* = \beta W$*.*

The clipped bounds maintain symmetry with respect to the range midpoint $(l + u)/2$. When $\beta = 1$, we have $l^* = l$ and $u^* = u$, recovering the original LVQ bounds without any clipping. When $\beta < 1$, the range contracts symmetrically from both ends.

Error Decomposition. The reconstruction error in DalQ comprises two independent components: clipping error from range truncation and quantization error from uniform scalar quantization within the effective range. DalQ's adaptive range tightening creates a principled trade-off between the two error components:

– **Reduced quantization error**: With a narrower effective range $W^* = \beta W$, the quantization step size becomes $\Delta^* = \beta\Delta$, which is smaller than LVQ's

step size by a factor of β. For dimensions with values inside the clipped range $[l^*, u^*]$, the quantization error reduces proportionally to β^2 (since error is proportional to Δ^2).
- **Controlled clipping error**: Dimensions with values outside the clipped range $[l^*, u^*]$ are clipped to the nearest boundary (l^* or u^*) before quantization. This introduces additional clipping error for outlier dimensions.

The fundamental insight is that for Gaussian-like distributions, the substantial reduction in quantization error affecting the high-density majority typically outweighs the controlled increase in clipping error affecting the low-density minority, yielding lower overall MSE. The optimal clip factor β^* for a given bit-width b precisely balances these two error components to minimize the total expected MSE.

Theorem 1 (Existence and Uniqueness of Optimal Clip Factor). *Suppose dimension values z_j within a vector are i.i.d. from a continuous unimodal distribution $p(z)$ with mode $m \in (l, u)$, i.e., p is non-decreasing on $(-\infty, m]$ and non-increasing on $[m, +\infty)$. Then for each bit-width $b \geq 1$, there exists a unique optimal clip factor $\beta^*(b) \in (0, 1]$ that minimizes the total MSE.*

Proof. For fixed b, define

$$f(\beta) \triangleq \mathrm{MSE}(\beta, b) = \varepsilon_{\mathrm{clip}}(\beta) + \varepsilon_{\mathrm{quant}}(\beta, b), \qquad \beta \in (0, 1],$$

where

$$\varepsilon_{\mathrm{clip}}(\beta) = \int_{-\infty}^{l^*(\beta)} (z - l^*(\beta))^2 p(z)\, dz + \int_{u^*(\beta)}^{+\infty} (z - u^*(\beta))^2 p(z)\, dz,$$

and

$$\varepsilon_{\mathrm{quant}}(\beta, b) = \frac{W^2 \beta^2}{12(2^b - 1)^2}.$$

We first prove existence. Define $f(0) \triangleq \lim_{\beta \to 0^+} f(\beta) = \mathbb{E}[(z - c)^2]$, where $c = (l + u)/2$. As $\beta \to 0^+$, the interval $[l^*(\beta), u^*(\beta)]$ shrinks to $\{c\}$, so the above extension is well defined. Hence, f is continuous on $[0, 1]$. Since $[0, 1]$ is compact, the Extreme Value Theorem implies that f attains its minimum at some $\hat{\beta} \in [0, 1]$. Because $\beta = 0$ is not optimal, there exists at least one minimizer $\beta^*(b) \in (0, 1]$.

We next prove uniqueness by showing that f is strictly convex on $(0, 1]$. For each fixed z, define $g(\beta; z) = \big(\max(z - u^*(\beta), 0)\big)^2 + \big(\max(l^*(\beta) - z, 0)\big)^2$. Since $l^*(\beta) = c - \beta W/2$ and $u^*(\beta) = c + \beta W/2$ are affine in β, and since $h(t) = \max(t, 0)^2$ is convex, it follows that $g(\beta; z)$ is convex in β. Therefore, $\varepsilon_{\mathrm{clip}}(\beta) = \mathbb{E}[g(\beta; z)]$ is convex. On the other hand, $\varepsilon_{\mathrm{quant}}(\beta, b) = \frac{W^2}{12(2^b - 1)^2}\beta^2$ is strictly convex because $\frac{d^2}{d\beta^2}\varepsilon_{\mathrm{quant}}(\beta, b) = \frac{W^2}{6(2^b - 1)^2} > 0$. Therefore, f is strictly convex on $(0, 1]$ as the sum of a convex function and a strictly convex function. A strictly convex function on a convex domain has at most one minimizer. Hence, the minimizer $\beta^*(b) \in (0, 1]$ exists and is unique.

Algorithm 1 Adaptive Search for Optimal Clip Factor

Input: Samples $\mathcal{S}$, bit-width b, range $[\beta_{\min}, \beta_{\max}]$, counts n_c, n_r, n_f, radii δ_r, δ_f
Output: Near-optimal β^*

$\mathcal{B}_c \leftarrow$ uniform sample of n_c values in $[\beta_{\min}, \beta_{\max}]$
$\beta_{\text{best}} \leftarrow \textsc{EvaluateCandidates}(\mathcal{B}_c, \mathcal{S}, b)$
$\mathcal{B}_r \leftarrow$ uniform sample of n_r values in $[\beta_{\text{best}} - \delta_r, \beta_{\text{best}} + \delta_r]$
$\beta_{\text{best}} \leftarrow \textsc{EvaluateCandidates}(\mathcal{B}_r, \mathcal{S}, b)$
$\mathcal{B}_f \leftarrow$ uniform sample of n_f values in $[\beta_{\text{best}} - \delta_f, \beta_{\text{best}} + \delta_f]$
$\beta_{\text{best}} \leftarrow \textsc{EvaluateCandidates}(\mathcal{B}_f, \mathcal{S}, b)$
return β_{best}

Adaptive Search Algorithm. While computing the optimal clip factor analytically is intractable, we adopt an adaptive search strategy to obtain a near-optimal solution β^*. Given quantization bit-width b and a sample set $\mathcal{S} = \{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$, we solve

$$\beta^*(b) = \arg \min_{\beta \in [\beta_{\min}, \beta_{\max}]} \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{z}_i - \hat{\mathbf{z}}_i(\beta, b)\|_2^2, \tag{4}$$

where $\hat{\mathbf{z}}_i(\beta, b)$ denotes the dequantized vector produced by DalQ under clip factor β and bit-width b. As shown in Algorithm 1, the search is performed in three stages. We first sample candidate clip factors uniformly over the full range $[\beta_{\min}, \beta_{\max}]$ to identify a promising region. We then progressively restrict the search to smaller neighborhoods centered at the current best candidate, using a refinement radius δ_r followed by a finer radius δ_f. At each stage, all candidates are evaluated by quantizing and dequantizing the sampled vectors and computing the corresponding average MSE. The candidate with the lowest reconstruction error is retained for the next stage. This coarse-to-fine design provides an efficient approximation to the optimal clip factor while avoiding exhaustive search over the entire range.

Besides, we identify a reference factor $\beta_{\text{ref}} = 1 - e^{-b}$ that provides a tuning-free approximation, enabling direct application or serving as effective initialization for the adaptive search. Detailed empirical validation and analysis are presented in Section 4.6.

3.2 Decomposed Vector Refinement

Beyond distribution-aware range adaptation, DalQ introduces a decomposed vector refinement framework (Algorithm 2) that exploits the geometric structure (Figure 3b) of vector quantization to achieve superior reconstruction accuracy. Classical quantization approaches, including LVQ, determine quantization codes purely based on value proximity—rounding each dimension to the nearest quantization level—without considering the overall geometric relationship between the original vector $\mathbf{z}$ and its reconstructed counterpart $\hat{\mathbf{z}}$. To minimize reconstruction error $\|\mathbf{z} - \hat{\mathbf{z}}\|^2$, we adopt a two-stage geometric optimization strategy:

Algorithm 2 Decomposed Vector Refinement

Input: Original vector $\mathbf{z}$, initial codes $\mathbf{c}$, clipped bounds l^*, u^*, bit-width b, iterations R

Output: Refined codes $\mathbf{c}'$, scale λ^*

// Stage 1: Directional Alignment via Iterative Code Search

1: $\mathbf{c}' \leftarrow \mathbf{c}$; $\hat{\mathbf{z}} \leftarrow \textsc{Decode}(\mathbf{c}', l^*, u^*, b)$; $\text{sim}_{\text{best}} \leftarrow \cos(\mathbf{z}, \hat{\mathbf{z}})$
2: **for** $r = 1$ to R **do**
3: **for** $i = 1$ to d **do**
4: Try searching code c'_i by $\delta \in \{-k, \ldots, -1, +1, \ldots, +k\}$
5: $\text{sim}_{\text{new}} \leftarrow \cos(\mathbf{z}, \hat{\mathbf{z}}_{\text{new}})$
6: **if** $\text{sim}_{\text{new}} > \text{sim}_{\text{best}}$ **then**
7: $c'_i \leftarrow c'_i + \delta$; $\text{sim}_{\text{best}} \leftarrow \text{sim}_{\text{new}}$
8: **end if**
9: **end for**
10: **end for**

// Stage 2: Magnitude Refinement via Projection-Based Scaling

11: $\hat{\mathbf{z}}_0 \leftarrow \textsc{Decode}(\mathbf{c}', l^*, u^*, b)$
12: $\lambda^* \leftarrow \langle \mathbf{z}, \hat{\mathbf{z}}_0 \rangle / \|\hat{\mathbf{z}}_0\|^2$
13: **return** $\mathbf{c}', \lambda^*$

first aligning the direction of the quantized vector with the original vector, then adjusting its magnitude to the optimal scale. This decomposition uses two sequential stages: directional alignment via iterative code search and magnitude refinement via projection-based scaling, as detailed below.

Directional Alignment via Iterative Code Search. The first stage of decomposed vector refinement (Algorithm 2, lines 1-10) focuses on finding quantization codes that maximize directional alignment between the original vector $\mathbf{z}$ and the reconstructed vector $\hat{\mathbf{z}}$. Directional alignment is measured by cosine similarity:

$$\text{sim}(\mathbf{z}, \hat{\mathbf{z}}) = \frac{\langle \mathbf{z}, \hat{\mathbf{z}} \rangle}{\|\mathbf{z}\| \cdot \|\hat{\mathbf{z}}\|} \tag{5}$$

However, given the bit-width b, there exists a discrete set of possible reconstructed vectors determined by the $(2^b)^d$ possible code combinations. Enumerating all possible combinations is computationally infeasible. Instead, we employ an iterative greedy search strategy.

Starting from the initial quantization codes obtained via standard rounding (Figure 3b, where the green square v_1 denotes the quantization codeword closest to v_0), we iteratively examine code modifications and accept those that improve directional alignment (Figure 3b, where the blue diamond v_2 denotes the codeword discovered after several search steps (blue dashed arrows) that is directionally closer to v_0 than v_1). The algorithm performs R rounds (line 2), where in each round we sweep through all dimensions (line 3). For each dimension i, we try searching the quantization code c'_i by small increments

$\delta \in \{-k, \ldots, -1, +1, \ldots, +k\}$ (line 4), compute the resulting cosine similarity (line 5), and accept improvements that increase cosine similarity (lines 6-7).

This greedy search efficiently explores the discrete code space to find codes that better preserve the directional alignment with the original vector. Since the procedure only accepts code modifications that strictly improve the cosine similarity between $\mathbf{z}$ and the decoded vector, the objective value increases monotonically throughout the search. Meanwhile, the code space is finite, as each dimension can only take one of 2^b discrete codes. Therefore, the search must terminate after a finite number of accepted updates. At convergence, no further code change within the prescribed local search neighborhood can improve the cosine similarity, indicating that the procedure reaches a stable coordinate-wise local optimum. Although this greedy strategy does not guarantee the global optimum over the entire discrete code space, it provides an effective and computationally lightweight mechanism for improving directional alignment in practice.

Magnitude Refinement via Projection-Based Scaling. After directional alignment fixes the quantization codes $\mathbf{c}'$, the corresponding decoded vector $\hat{\mathbf{z}}_0 = \textsc{Decode}(\mathbf{c}', l^*, u^*, b)$ determines a specific direction in $\mathbb{R}^d$ (Algorithm 2, line 11). However, the magnitude of $\hat{\mathbf{z}}_0$ may not optimally match the magnitude of the original vector $\mathbf{z}$. The key insight is that, given a fixed direction $\hat{\mathbf{z}}_0$, the reconstruction that minimizes mean squared error should lie along this direction at the point closest to $\mathbf{z}$.

Optimal Scaling Scalar. We seek a scalar λ^* such that the final reconstruction $\hat{\mathbf{z}} = \lambda^* \hat{\mathbf{z}}_0$ minimizes the mean squared error:

$$\lambda^* = \arg\min_{\lambda \in \mathbb{R}} \|\mathbf{z} - \lambda \hat{\mathbf{z}}_0\|^2 \tag{6}$$

Taking the derivative with respect to λ and setting it to zero:

$$\frac{d}{d\lambda} \|\mathbf{z} - \lambda \hat{\mathbf{z}}_0\|^2 = -2\langle \mathbf{z}, \hat{\mathbf{z}}_0 \rangle + 2\lambda \|\hat{\mathbf{z}}_0\|^2 = 0 \tag{7}$$

Solving for λ yields the optimal scaling scalar:

$$\lambda^* = \frac{\langle \mathbf{z}, \hat{\mathbf{z}}_0 \rangle}{\|\hat{\mathbf{z}}_0\|^2} \tag{8}$$

The optimal scaling scalar ensures that the final reconstructed vector $\hat{\mathbf{z}} = \lambda^* \hat{\mathbf{z}}_0$ achieves the minimum possible MSE along the direction determined by the quantization codes (Algorithm 2, line 12). Geometrically, λ^* scales $\hat{\mathbf{z}}_0$ such that $\hat{\mathbf{z}}$ is the orthogonal projection of $\mathbf{z}$ onto the ray emanating from the origin in the direction of $\hat{\mathbf{z}}_0$ (Figure 3b, where the yellow diamond v_3 denotes the point where v_0 is projected onto the direction of v_2).

Algorithm 3 DalQ Quantization

Input: Centered vector $\mathbf{z} \in \mathbb{R}^d$; bit-width b; clip factor β; refinement rounds R.
Output: Quantization codes $\mathbf{c} \in \{0, \ldots, 2^b - 1\}^d$; scale s; bias t.

// Step 1: Compute per-vector bounds and apply clipping
1: $l \leftarrow \min_{j\in[d]} z_j$; $u \leftarrow \max_{j\in[d]} z_j$; $W \leftarrow u - l$
2: $l^* \leftarrow l + \frac{1-\beta}{2}W$; $u^* \leftarrow u - \frac{1-\beta}{2}W$
// Step 2: Initial quantization via rounding
3: **for** $j = 1$ to d **do**
4: $\quad z'_j \leftarrow \max(l^*, \min(z_j, u^*))$
5: $\quad c_j \leftarrow \left\lfloor \frac{z'_j - l^*}{u^* - l^*}(2^b - 1) + 2^{-1} \right\rfloor$
6: **end for**
// Step 3: Decomposed vector refinement
7: $(\mathbf{c}, \lambda^*) \leftarrow \textsc{DecomposedVectorRefinement}(\mathbf{z}, \mathbf{c}, l^*, u^*, b, R)$
// Step 4: Compute final scale and bias
8: $s \leftarrow \frac{u^* - l^*}{2^b - 1} \cdot \lambda^*; t \leftarrow l^* \cdot \lambda^*$
9: **return** $\mathbf{c}, s, t$

Algorithm 4 DalQ Dequantization

Input: Quantization codes $\mathbf{c} \in \{0, \ldots, 2^b - 1\}^d$; scale s; bias t.
Output: Reconstructed vector $\hat{\mathbf{z}} \in \mathbb{R}^d$.

// Dequantize each dimension via affine transformation
1: **for** $j = 1$ to d **do**
2: $\quad \hat{z}_j \leftarrow t + c_j \cdot s$
3: **end for**
4: **return** $\hat{\mathbf{z}}$

3.3 Quantization and Dequantization Procedure

We now present the complete DalQ procedures, integrating distribution-aware range adaptation and decomposed vector refinement.

Algorithm 3 presents the complete DalQ quantization procedure. The algorithm proceeds in four steps. Step 1 (lines 1-2) computes the per-vector bounds and applies the clip factor to obtain clipped bounds l^*, u^*. Step 2 (lines 3-6) performs initial quantization via standard rounding: for each dimension, clip to the effective range and map to a quantization code. Step 3 (line 7) applies decomposed vector refinement via Algorithm 2, which performs directional optimization via iterative code search and magnitude optimization via projection-based scaling, returning the refined codes $\mathbf{c}$ and the optimal projection scalar λ^*. Step 4 (line 8) computes the final scale and bias parameters that incorporate both the clipped range and the magnitude correction factor. Algorithm 4 presents the DalQ dequantization procedure. The dequantization procedure is efficient. Each dimension is reconstructed via an affine transformation $\hat{z}_j = t + c_j \cdot s$ (line 2), where the scale s and bias t were precomputed during quantization. DalQ stores the following for each vector: (1) quantization codes $\mathbf{c}$: $b \cdot d$ bits; (2) scale s: 4

bytes (32-bit float); (3) bias t: 4 bytes (32-bit float). The total storage is $b \cdot d + 64$ bits.

Distance Computation. DalQ employs a reconstruction-based distance computation approach, which first dequantizes the compressed representation to reconstruct the approximate vector, then computes distances in the original floating-point space. Critically, DalQ performs fused decode-and-compute operations at the register level—low-bit integer codes are decoded directly into CPU registers, and the resulting floating-point values remain resident in registers for immediate distance computation, eliminating unnecessary memory writes and enabling efficient SIMD-based vectorization.

Integration with ANN Indexing Systems. DalQ seamlessly integrates with ANN indexing systems as a drop-in replacement for LVQ or other scalar quantization methods. During index construction, each data vector $\mathbf{x}_i$ is quantized via Algorithm 3, producing a compressed representation $(\mathbf{c}_i, s_i, t_i)$ that replaces the original high-dimensional vector in memory. During query processing, to estimate distance between the query and a candidate, the candidate is on-the-fly dequantized via Algorithm 4 using its stored compressed representation, and the distance is computed in the original space. The fused decode-compute operations on compressed quantization codes enable efficient distance computation, thereby achieving DalQ's superior recall-QPS trade-offs in practical ANN search tasks.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on six publicly available large-scale datasets spanning diverse domains and dimensionalities (Table 1). These datasets include widely adopted benchmarks for the evaluation of ANN algorithms [11,10] (DEEP, MSong and GIST[1]) and embeddings generated by advanced AI models (YouTube[2], OpenAI-1536[3] and OpenAI-3072[4]). For the YouTube dataset, we randomly sampled 1 million vectors from the entire collection for evaluation. We exclude 1,000 vectors from each dataset and use them as the query vectors.

Baselines. (1) SQ [8]: The classical scalar quantization method implemented in Faiss utilizes global min-max bounds. (2) PQ [17]: A representative codebook-based method that achieves high compression through subspace quantization.

[1] https://www.cse.cuhk.edu.hk/systems/hash/gqr/datasets.html

[2] https://research.google.com/youtube8m/download.html

[3] https://huggingface.co/datasets/Qdrant/dbpedia-entities-openai3-text-embedding-3-large-1536-1M

[4] https://huggingface.co/datasets/Qdrant/dbpedia-entities-openai3-text-embedding-3-large-3072-1M

Table 1: Dataset Statistics

Dataset	Size	Dim	Query Size	Type
DEEP	1,000,000	256	1000	Image
MSong	994,185	420	1000	Audio
GIST	1,000,000	960	1000	Image
YouTube	1,000,000	1024	1000	Video
OpenAI-1536	1,000,000	1536	1000	Text
OpenAI-3072	1,000,000	3072	1000	Text

(3) LVQ [1]: A state-of-the-art scalar quantization method that employs per-vector min-max bounds. (4) RaBitQ [10]: A state-of-the-art scalar quantization method that operates on normalized vectors to achieve high accuracy.

Performance Metrics and Parameter Settings. For *quantization accuracy*, we measure mean squared error (MSE) normalized by vector dimensionality: $\text{MSE} = \frac{1}{nd}\sum_{i=1}^{n}\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2$. For *quantization efficiency*, we record the time required to quantize the entire dataset using 12 threads. For *distance computation efficiency*, we measure the latency of computing the Euclidean distance between a raw query vector and a quantized data vector in a single-threaded setting. For *ANNS performance*, we measure Recall@100 and queries per second (QPS). We integrate quantization methods with IVF indexing using FAISS [8]. Following common practice [10], we configure IVF with 4096 clusters and conduct single-threaded evaluation. In all experiments, we use the default parameters for DalQ: clip factor $\beta = 1 - e^{-b}$ and refinement rounds $R = 4$. We do not apply random rotation as a preprocessing step; all results are therefore obtained on the original datasets. A systematic evaluation of the impact of random rotation on DalQ's quantization accuracy and efficiency is left for future work. All methods were implemented in C++ using AVX-512 SIMD optimization. Experiments were conducted on a server equipped with 2 AMD EPYC 9554 CPUs, 768 GB RAM, and running Ubuntu 22.04. The source code is available at https://github.com/zhenghao24/DalQ.

4.2 Quantization Accuracy

We evaluate quantization accuracy by MSE, comparing DalQ against two state-of-the-art scalar quantization baselines: LVQ and RaBitQ. Experiments are conducted at bit-widths of 1, 2, 4, and 8, which are the most widely adopted settings in real-world quantization deployments.

As shown in Table 2, DalQ consistently achieves lower MSE than both LVQ and RaBitQ across all bit-width configurations. Compared to LVQ, the most substantial gains occur at 1-bit quantization (up to 96.6% reduction), where range adaptation and decomposed vector refinement work together to handle the highly constrained bit budget. The improvement decreases steadily as bit-width

Table 2: MSE comparison. We report absolute MSE values and relative MSE reduction achieved by DalQ compared to LVQ and RaBitQ.

Method	DEEP				MSong			
	b=1	b=2	b=4	b=8	b=1	b=2	b=4	b=8
LVQ	6.0e−3	4.0e−4	1.6e−5	5.5e−8	1.0e1	6.2e−1	3.2e−2	1.1e−4
RaBitQ	1.4e−3	1.9e−4	1.7e−5	6.1e−8	7.2e−1	3.2e−1	3.8e−2	1.7e−4
DalQ	5.0e−4	1.6e−4	1.4e−5	5.5e−8	4.1e−1	2.0e−1	2.5e−2	1.1e−4
vs. LVQ (%)	**91.8**	**60.1**	**14.2**	**0.7**	**96.0**	**67.5**	**20.9**	**0.5**
vs. RaBitQ (%)	**63.4**	**16.0**	**20.0**	**10.0**	**43.6**	**38.1**	**32.9**	**36.3**
Method	GIST				YouTube			
	b=1	b=2	b=4	b=8	b=1	b=2	b=4	b=8
LVQ	5.7e−3	4.5e−4	1.9e−5	6.7e−8	6.9e−1	4.4e−2	1.7e−3	6.0e−6
RaBitQ	9.6e−4	1.6e−4	2.1e−5	1.1e−7	1.1e−1	1.5e−2	1.3e−3	6.6e−6
DalQ	4.0e−4	1.5e−4	1.5e−5	6.7e−8	4.1e−2	1.3e−2	1.3e−3	6.0e−6
vs. LVQ (%)	**93.0**	**66.4**	**24.9**	**0.1**	**94.2**	**69.7**	**27.2**	**0.3**
vs. RaBitQ (%)	**58.5**	**7.9**	**29.2**	**38.4**	**63.1**	**11.5**	**4.6**	**8.9**
Method	OpenAI-1536				OpenAI-3072			
	b=1	b=2	b=4	b=8	b=1	b=2	b=4	b=8
LVQ	9.1e−4	5.7e−5	2.2e−6	7.6e−9	2.4e−3	1.5e−4	5.4e−6	1.9e−8
RaBitQ	1.3e−4	1.8e−5	1.5e−6	8.4e−9	2.1e−4	3.3e−5	3.2e−6	2.1e−8
DalQ	4.8e−5	1.6e−5	1.5e−6	7.6e−9	8.3e−5	2.9e−5	3.2e−6	1.8e−8
vs. LVQ (%)	**94.8**	**72.1**	**30.0**	**0.3**	**96.6**	**80.2**	**40.7**	**0.7**
vs. RaBitQ (%)	**62.9**	**11.4**	**1.6**	**9.1**	**60.1**	**10.6**	**0.5**	**10.4**

increases, declining from over 90% at 1 bit to below 1% at 8 bits. We attribute this trend to two factors: first, the optimal clip factor approaches 1 at higher bit-widths, rendering range adaptation less effective; second, the per-vector bounds in LVQ already yield high quantization accuracy at wider bit-widths, leaving limited room for decomposed vector refinement to further improve upon. Compared to RaBitQ, DalQ achieves up to 63.4% MSE reduction at 1 bit and maintains a consistent advantage across all bit-width settings. Notably, at 8 bits, both LVQ and DalQ outperform RaBitQ, indicating that the use of per-vector quantization ranges becomes increasingly advantageous over fixed-range methods at higher bit-widths. Overall, the results demonstrate that DalQ achieves state-of-the-art quantization accuracy among existing scalar quantization methods.

4.3 Quantization Efficiency

Table 3 reports the encoding time (in seconds) for each dataset at different bit-widths. Both LVQ and DalQ take roughly the same time to encode, regardless of

Table 3: Quantization time (in seconds) of different methods across datasets with varying dimensionalities. *Speedup* denotes the speedup of DalQ over RaBitQ.

Method	DEEP			MSong			OpenAI-1536			OpenAI-3072		
	$B=2$	$B=4$	$B=8$	$B=2$	$B=4$	$B=8$	$B=2$	$B=4$	$B=8$	$B=2$	$B=4$	$B=8$
LVQ	0.2	0.3	0.3	0.5	0.5	0.6	2.3	2.5	2.6	4.1	4.3	4.4
RaBitQ	1.6	3.1	10.7	3.1	5.7	18.9	11.3	23.2	70.3	24.4	41.9	167.9
DalQ	0.5	0.6	0.6	0.9	1.1	1.2	3.5	3.8	3.7	6.1	6.4	6.5
Speedup	**3.2×**	**5.2×**	**17.8×**	**3.4×**	**5.2×**	**15.8×**	**3.2×**	**6.1×**	**19.0×**	**4.0×**	**6.5×**	**25.8×**

bit-width. DalQ is slightly slower than LVQ due to the extra decomposed refinement step. In contrast, RaBitQ's encoding time grows sharply with bit-width, reaching 167.9 s on OpenAI-3072 at $B = 8$, compared to only 6.5 s for DalQ. As a result, DalQ's speedup over RaBitQ grows with bit-width, reaching up to 25.8× on OpenAI-3072 at $B = 8$. The combination of near-LVQ efficiency and superior accuracy makes DalQ particularly attractive for large-scale deployment scenarios where both quantization quality and computational efficiency are critical.

4.4 Distance Computation Efficiency

We measure the time to compute the Euclidean distance between one query vector and one quantized data vector. For each dataset, we sample 10,000 data vectors and 10 query vectors and report the average time per pair. LVQ is excluded because it uses the same computation strategy as DalQ and achieves similar speed. Since the computation time is nearly the same across bit-widths for both methods, Table 4 reports results under 8-bit quantization. DalQ outperforms RaBitQ in distance computation across all datasets, achieving speedups of up to 3.2×. The speedup comes from DalQ's fused decode-and-compute design: integer codes are decoded directly into registers and used immediately for distance computation, avoiding extra memory accesses. RaBitQ uses a stepwise decomposition approach that requires more intermediate operations, and the overhead grows with vector dimensionality.

Table 4: Euclidean distance computation time (in nanoseconds) under 8-bit quantization. *Speedup* denotes the speedup of DalQ over RaBitQ.

Method	DEEP	MSong	GIST	YouTube	OpenAI-1536	OpenAI-3072
RaBitQ	67	85	122	131	189	323
DalQ	21	30	55	58	92	158
Speedup	**3.2×**	**2.8×**	**2.2×**	**2.3×**	**2.1×**	**2.0×**

4.5 ANNS Performance

We evaluate DalQ's Approximate Nearest Neighbor Search (ANNS) performance by measuring the recall-QPS trade-off when integrated with standard IVF indexing systems. As depicted in Figure 4, under 8-bit quantization across four datasets, DalQ consistently outperforms baselines including SQ, PQ, and LVQ.

Compared to the state-of-the-art RaBitQ, DalQ demonstrates highly competitive overall ANNS performance. At moderate recall levels ($\leq 90\%$), DalQ achieves a higher QPS, while the performance of the two methods converges as the target recall increases. This behavior stems from their distinct algorithmic designs: DalQ employs a dimensional pruning strategy that utilizes leading dimensions for distance estimation, whereas RaBitQ achieves more effective pruning based on distance estimations derived from single-bit quantization. Nevertheless, DalQ bridges this pruning gap through vastly superior distance computation speeds. Furthermore, DalQ's quantization process is tens of times faster, making it well-suited for end-to-end scenarios that demand rapid index construction and high throughput, such as real-time streaming ingestion, on-the-fly indexing, and massive-scale database updates. For example, during an end-to-end ANN search on the openai-1536 dataset, DalQ can process over 740,000 queries on a single thread before RaBitQ even begins to respond. These results also indicate that further optimizing DalQ's pruning strategy is a promising research direction.

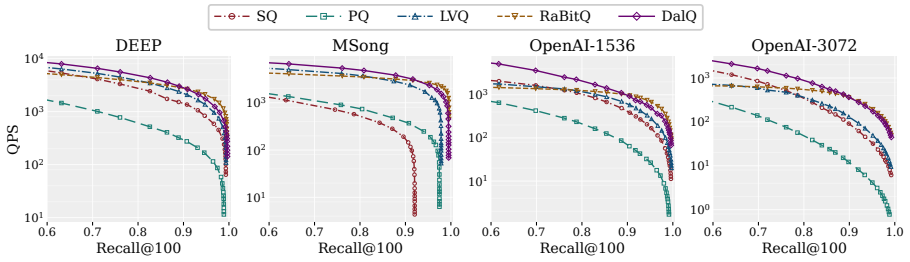


Fig. 4: ANNS performance under 8-bit quantization. Curves closer to the top-right corner demonstrate better performance.

4.6 Characterization of Clip Factor

We characterize the optimal clip factor $\beta^*(b)$ across six benchmark datasets for $b \in [1, 8]$ using the adaptive search algorithm (see supplementary material). Figure 5 shows the measured optima and fitted curves.

Sigmoid Characterization and Dimensionality Effect. On all six datasets, $\beta^*(b)$ is well described by a sigmoid curve: $\beta^*(b) \approx \frac{\alpha}{1+\exp(-\gamma(b-\delta))} + \epsilon$, ($R^2 >$

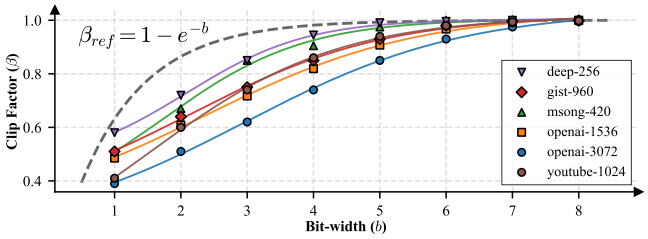


Fig. 5: Empirical optimal clip factors (data points) and fitted curves. The black dashed curve represents the reference formula $\beta_{\text{ref}} = 1 - e^{-b}$.

0.98), exhibiting a three-phase behavior: small values at low b (aggressive clipping compensates for coarse quantization), a smooth transition region, and saturation at large b (finer quantization steps render aggressive clipping unnecessary). Furthermore, for a fixed bit-width, higher-dimensional datasets consistently yield smaller $\beta^*(b)$, as higher dimensionality stabilizes per-vector distributional statistics and supports stronger range reduction.

Reference Factor. We further identify a reference curve $\beta_{\text{ref}} = 1 - e^{-b}$ that lies above all measured optima on all six datasets, providing a safe, tuning-free default for deployment.

5 Related Work

5.1 Vector Quantization

Product Quantization and Variants. PQ [17] decomposes vectors into subspaces and quantizes each independently via learned codebooks, enabling asymmetric distance computation and favorable recall-efficiency trade-offs. Subsequent methods extend this foundation: OPQ [13] learns rotation matrices to minimize quantization error; AQ [3] represents vectors as sums of multiple codewords for finer approximations; and LSQ++ [20] combines multiple codebooks with locally-adaptive scaling. Despite their high compression ratios, all these methods share a common bottleneck: expensive codebook training, memory-intensive storage, and table-lookup-based distance computation create scalability barriers in large-scale deployments.

Scalar Quantization and Variants. SQ quantizes each dimension independently without codebook learning, offering significant practical advantages over PQ-based methods: no training overhead, no codebook storage, and efficient SIMD-based distance computation without table lookups. Classical SQ [8] computes dataset-wide bounds but underutilizes quantization ranges for individual vectors. LVQ [1] introduces a per-vector quantization range by computing individual bounds for each vector, thereby improving range utilization. Ra-

BitQ [11,10] introduces an unbiased estimator and achieves asymptotically optimal error bounds, but suffers from poor quantization efficiency at high bit-widths and limits distance computation to Euclidean distance and inner product only.

5.2 Approximate Nearest Neighbor Search

Approximate nearest neighbor (ANN) search [15,23] is a fundamental problem in high-dimensional data management, with applications spanning recommendation systems [12,5], information retrieval [26,21], and retrieval-augmented generation [18]. Existing ANN approaches fall into several families: tree-based methods [6], hashing-based methods [7,25], graph-based methods [19,2], and subspace-based methods [24]. Modern vector database systems [22,14] integrate these indexing structures with quantization-based compression to enable billion-scale similarity search with memory efficiency.

6 Conclusions

In this paper, we presented DalQ (Distribution-aware locally-adaptive Quantization), a vector quantization method that achieves high accuracy and computational efficiency through distribution-aware range adaptation and decomposed vector refinement. Extensive experiments across diverse large-scale benchmarks demonstrate that DalQ consistently outperforms existing baselines in both compression quality and efficiency. As a drop-in replacement for vector quantization, DalQ integrates seamlessly with standard ANN indexing frameworks, making it well-suited for large-scale production deployments.

Acknowledgments. This work is supported by Huawei(TC20250520001) and partially supported by EU Horizon projects ARMADA (101168951), TwinODIS (101160009), DataGEMS (101188416), $\Upsilon\Pi AI\Theta A$ & NextGenerationEU project HARSH ($\Upsilon\Pi 3TA-$ 0560901), DEDALUS CREs (TA 5180519). This work was granted access to the HPC resources of IDRIS under the allocation 2025-A0191012641 made by GENCI.

Declaration of Generative AI Use. The authors used Google Gemini only for grammar checking and language polishing of the manuscript. It was not used to generate research ideas, scientific content, analyses, figures, or experimental results. All scientific content and conclusions in this paper are the sole work of the authors.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Aguerrebere, C., Bhati, I.S., Hildebrand, M., Tepper, M., Willke, T.: Similarity search in the blink of an eye with compressed indices. VLDB (2023)
2. Azizi, I., Echihabi, K., Palpanas, T.: Elpis: Graph-based similarity search for scalable data science. VLDB (2023)

3. Babenko, A., Lempitsky, V.: Additive quantization for extreme vector compression. In: CVPR (2014)
4. Billingsley, P.: Probability and Measure. Wiley, 3rd edn. (1995)
5. Bin, X., Cui, J., et al.: Real-time indexing for large-scale recommendation by streaming vector quantization retriever. In: SIGKDD (2025)
6. Dasgupta, S., Freund, Y.: Random projection trees and low dimensional manifolds. In: STOC (2008)
7. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: SoCG (2004)
8. Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.E., Lomeli, M., Hosseini, L., Jégou, H.: The faiss library. arXiv (2024)
9. Frantar, E., Ashkboos, S., Hoefler, T., Alistarh, D.: Gptq: Accurate post-training quantization for generative pre-trained transformers. arXiv (2022)
10. Gao, J., Gou, Y., Xu, Y., Yang, Y., Long, C., Wong, R.C.W.: Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search. SIGMOD (2025)
11. Gao, J., Long, C.: Rabitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. SIGMOD (2024)
12. Gao, Y., Xiong, Y., Gao, X., Jia, K., et al.: Retrieval-augmented generation for large language models: A survey. arXiv (2023)
13. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. In: CVPR (2013)
14. Guo, R., Luan, X., Xiang, L., Yan, X., Yi, X., Luo, J., Cheng, Q., Xu, W., et al.: Manu: a cloud native vector database management system. VLDB (2022)
15. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC (1998)
16. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML (2015)
17. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence (2010)
18. Jeong, T.: 4bit-quantization in vector-embedding for rag. In: ICMLA (2024)
19. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. TPAMI (2018)
20. Martinez, J., Zakhmi, S., Hoos, H.H., Little, J.J.: Lsq++: Lower running time and higher recall in multi-codebook quantization. In: ECCV (2018)
21. O'Neill, J., Dutta, S.: Improved vector quantization for dense retrieval with contrastive distillation. In: SIGIR (2023)
22. Wang, J., Yi, X., Guo, R., Jin, H., Xu, P., Li, S., Wang, X., Guo, X., et al.: Milvus: A purpose-built vector data management system. In: SIGMOD (2021)
23. Wang, Z., Xiong, H., Wang, Q., He, Z., Wang, P., Palpanas, T., Wang, W.: Dimensionality-reduction techniques for approximate nearest neighbor search: A survey and evaluation. IEEE Data Eng. Bull. (2024)
24. Wei, J., Lee, X., et al.: Subspace collision: An efficient and accurate framework for high-dimensional approximate nearest neighbor search. SIGMOD (2025)
25. Wei, J., Peng, B., Lee, X., et al.: Det-lsh: A locality-sensitive hashing scheme with dynamic encoding tree for approximate nearest neighbor search. VLDB (2024)
26. Xiao, S., Liu, Z., Han, W., et al.: Distill-vq: Learning retrieval oriented vector quantization by distilling knowledge from dense embeddings. In: SIGIR (2022)