JHQ: Johnson-Lindenstrauss Enhanced Hierarchical Quantization for High-Dimensional Approximate Nearest Neighbor Search

Jiabao Han
Australian National University
Canberra, Australia
Jiabao.Han@anu.edu.au

Mengxuan Zhang
Australian National University
Canberra, Australia
Mengxuan.Zhang@anu.edu.au

Goce Trajcevski
Iowa State University
Ames, United States
gocet25@iastate.edu

ABSTRACT

High-dimensional approximate nearest neighbor (ANN) search provides efficiency and scalability that are fundamental to modern AI applications such as retrieval-augmented generation and recommendation systems. While vector quantization (VQ) methods excel at compressing vectors for efficient search, existing approaches face critical bottlenecks: (i) prolonged indexing times due to expensive data-dependent training; (ii) slow query processing from quadratic distance computations; and (iii) poor scalability on large datasets. In this paper, we introduce a novel quantization framework that leverages the orthogonal Johnson-Lindenstrauss (JL) transformation to lay the foundation for resolving these bottlenecks. Our key insight is that the JL transformation induces a predictable near-Gaussian distribution with independent dimensions, enabling quick codebook generation without expensive iterative training. Based on this, we propose two algorithms: JQ (JL-enhanced Quantization) achieves fast indexing through training-free codebook construction while maintaining provable distance error bounds; and JHQ (JL-enhanced Hierarchical Quantization) extends JQ with a two-level architecture that uses primary quantization for rapid candidate filtering and residual quantization for accurate refinement, achieving a better query accuracy–speed trade-off on large-scale datasets. Finally, extensive experiments on six benchmark datasets with up to 3,072 dimensions demonstrate that our methods achieve 310× query speedups over state-of-the-art baselines at ≥95% recall, with 10–30× index construction speedups. In particular, JHQ excels on massive datasets, maintaining 210× higher queries per second at >90% recall compared to JQ.

PVLDB Reference Format:
Jiabao Han, Mengxuan Zhang, and Goce Trajcevski. JHQ: Johnson-Lindenstrauss Enhanced Hierarchical Quantization for High-Dimensional Approximate Nearest Neighbor Search. PVLDB, 19(7): 1530 - 1543, 2026.
doi:10.14778/3801059.3801067

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/jiabhan/JHQ.

Proceedings of the VLDB Endowment, Vol. 19, No. 7 ISSN 2150-8097.
doi:10.14778/3801059.3801067

* Mengxuan Zhang is the corresponding author.

1 INTRODUCTION

Nearest neighbor (NN) search is a fundamental operation in high-dimensional spaces and is at the heart of applications such as retrieval-augmented generation [23, 37], information retrieval [35, 54], and recommendation systems [55]. However, exact NN search is computationally prohibitive in high-dimensional spaces due to the *curse of dimensionality* [10, 11, 31, 68]. In addition, the ever-increasing data dimensionality may often result in memory bandwidth constraints during the search [1]. In order to cater to realistic constraints and achieve a better query accuracy-efficiency trade-off, researchers and practitioners have turned to the approximate nearest neighbor (ANN) search, a paradigm which naturally emerged as a practical alternative, yielding significant processing gains at the expense of an acceptable error in accuracy. The ANN works can be classified into four main categories: (1) Locality-Sensitive Hashing (LSH)-based [19, 30, 58, 60]; (2) Vector Quantization (VQ)-based [21, 38, 50, 67]; (3) Tree-based [16, 45, 53, 56]; and (4) graph-based [17, 28, 40, 41] approaches. Among them, VQ is particularly important since it excels at alleviating the memory bandwidth pressure by compressing high-dimensional vectors into compact codes, achieving high query throughput. However, its accuracy is relatively low. In this paper, we present a novel VQ-based approach with higher accuracy, while maintaining the advantages in query processing.

Over the years, a plethora of VQ-based ANN algorithms have been proposed to improve query accuracy. For instance, Product Quantization (PQ) [33] compresses vectors by partitioning the vector space into subspaces, learns a codebook for each subspace, and uses the learned codebooks for fast table lookup in query processing. However, PQ relies on the assumption of subspace independence, which, as a consequence, leads to low accuracy on correlated data [7, 24, 42]. Its optimized variants [24, 47] propose to reduce the correlation by training an orthogonal matrix, but they also introduce $O(d^3)$ training overhead (where d indicates the dimensionality). As an alternative, Residual Vector Quantization (RVQ) [13] removes the assumption by operating in full-dimensional space. Specifically, it hierarchically approximates vectors with full-dimensional codebooks, with each subsequent level refining the quantization error of the previous one. However, it suffers from diminishing returns and intensive codebook training due to degrading residual entropy [38]. Additive Quantization (AQ) and its improvements [7, 8, 42, 43, 72] further relax RVQ's sequential constraint and represent vectors with multiple interdependent codebooks, achieving higher accuracy, but at the cost of NP-hard encoding and quadratic query complexity. Other RVQ variants [38, 50] demonstrate a potential to surpass AQ approaches through sophisticated optimizations but remain training-intensive [50].

Observations. Despite various attempts to improve the accuracy of VQ-based methods, there remain three important bottlenecks:

B1. *Increased indexing time:* Recent advances aim to reduce quantization error by relaxing the subspace independence assumption, but they typically involve computationally intensive training procedures that significantly prolong index construction time;

B2. *Slower query processing:* To boost accuracy, both RVQ- and AQ-based methods use multiple full-dimensional codebooks. This requires calculations between pairs of codebooks to encode each vector, making the query processing quadratically slower than the simple codebook lookup;

B3. *Scalability issue with large datasets:* As the dataset size grows, vectors are more likely to be quantized to the same code, which could reduce query accuracy. But if we prolong the quantization code for more accurate query processing, it will slow down query speed, adversely affecting the query performance aspect of the trade-off on large datasets.

Essentially, existing VQ-based methods boost query accuracy with extensive codebook training to adapt to the underlying data distribution via iterative optimization. Since the data distribution is unknown beforehand and varies across different datasets, the search space for optimal codebooks is enormous, making the problem NP-hard [7]. This naturally raises the question: can we improve the indexing time by first transforming the dataset into one with a known distribution? This would enable the codebook to be computed more efficiently, without time-consuming training. Complementarily, as the query processing of existing VQ-based methods is slowed down by expensive distance computation with complex codebooks, we can achieve faster query processing if the codebooks can be constructed for fast distance calculation. Lastly, to improve the trade-off between the query speed and accuracy in large-scale datasets, how about quickly filtering the candidates with short codes and refining them with more precise quantization code for higher accuracy?

Challenges. Towards exploring the potential of this idea and overcoming the three bottlenecks, we identify the following challenges and, along the way, highlight the main ideas of our approach:

Challenge 1: How to keep index construction time low while maintaining high recall? As analyzed, it is possible to construct an index quickly by getting rid of the expensive training, by transforming the datasets into a determined or known distribution. To make this idea feasible, the transformation is required to satisfy two conditions: First, it must preserve the distance ranking, i.e., for any vector, its top-k nearest neighbors remain the same after transformation, so as to maintain the query accuracy; Second, the transformation should be fast enough and not to bring significant additional computational overhead, so that the saved time will not be canceled out. Having these requirements in mind, we discover a nicely matched approach, the *orthogonal Johnson–Lindenstrauss (JL)* transformation, whose properties are detailed in Section 3. Importantly, we find that it transforms so that it follows an identical, near-Gaussian distribution. That means we can use and share one scalar (i.e., one-dimensional) codebook among all dimensions for quantization. As a result, this strategy successfully resolves B1 since it can largely reduce indexing time by getting rid of training overhead. It can also achieve high query accuracy with near-optimal quantization error, but its query processing is even slower than exhaustive search.

Challenge 2: With the fast index construction, can we continue to accelerate the query processing? The JL-based strategy essentially obtains the gain in indexing time and query accuracy at the cost of query speed. To further improve the performance aspect of the trade-off, we need to seek an approach that retains the benefits of JL transformation while improving the query efficiency, which is essentially constrained by the length of the quantization code. Therefore, instead of quantizing each dimension separately, we propose our JL-enhanced vector quantization (JQ) algorithm by dividing the full dimensionality into multiple small subspaces and applying a compact vector quantizer to each subspace. In this way, the quantization code can be shorter, which would support faster query answering and help to resolve B2. We discover that the subspace independence holds right after the orthogonal JL transformation, and the subspace dependence is precisely the root cause of various designs to boost the query accuracy in existing methods. So our JQ method can potentially achieve high accuracy, and we prove that its distance distortion error is bounded.

Challenge 3: How to maintain the query performance in large-scale datasets? Although our JQ algorithm achieves both high query speed and accuracy, its performance could degrade when the dataset size becomes larger, since it would be more difficult to discriminate between vectors with quantization codes of the same number of bits. We can maintain the query accuracy by prolonging the quantization code, but it will slow down the query processing. To resolve the conflict between query speed and accuracy with large datasets, we design a hierarchical quantization architecture (JHQ) by integrating quantization codes with different levels of precision. To be specific, we leverage short codes with relatively low quantization precision to quickly select a small candidate set. Then, a quantization with high precision is leveraged in the small set to obtain the final query results with high accuracy. This hierarchical design delivers the best of both worlds: high query processing speed with the coarse search quickly pruning the search space, while high accuracy with the long quantization code applied to the candidates. Moreover, the candidate set size is tunable, which supports a flexible trade-off between query speed and accuracy.

Our contributions can be summarized as follows:

(1) We identify the bottlenecks (cf. B1, B2, and B3) in existing VQ-based ANN algorithms and discover a vector transformation method, JL, that can potentially resolve them.
(2) We design a scalar quantization solution with fast index construction (cf. B1) by proving the distance preservation and leveraging the dimensional independence property of JL.
(3) To achieve faster query processing (cf. B2), we propose a novel JL-enhanced algorithm JQ by quantizing vectors with compact codes, and prove that JQ can maintain high accuracy with bounded distance error.
(4) To deal with the scalability issue (cf. B3), we put forward a hierarchical quantization architecture JHQ that enables a flexible trade-off between query speed and accuracy.
(5) We conduct extensive experiments on six large-scale, high-dimensional benchmark datasets. The results demonstrate that our algorithms JQ and JHQ can achieve the best query performance in most datasets with much faster index construction, compared to the state-of-the-art baselines.

We note that due to space constraints, the proofs of the Lemmas and Theorems, along with additional experiments supporting the quantitative observations, have been moved to an Appendix, which is publicly available at https://github.com/jiabhan/JHQ.

2 PRELIMINARIES

We now formally define the ANN search problem and review the key concepts of vector quantization that form the basis of our work.

2.1 Problem Setting

Let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^{d \times n}$ ($n \in \mathbb{N}^+$) denote a database of n points with dimensionality d. Given a query point $\mathbf{q} \in \mathbb{R}^d$, the goal of k-Nearest Neighbor (kNN) search is to find the set $\mathcal{N}_k(\mathbf{q}) \subset \mathcal{X}$ containing the top-k closest points to $\mathbf{q}$, based on a given distance function $d(\cdot, \cdot)$. As exact kNN search is computationally infeasible in high dimensions, practical systems rely on Approximate kNN (ANN) search. This approach returns an approximate result set $\mathcal{N}'_k(\mathbf{q})$, where $\forall \mathbf{x}' \in \mathcal{N}'_k(\mathbf{q})$, it satisfies $d(\mathbf{q}, \mathbf{x}') \leq (1+\varepsilon) d(\mathbf{q}, \mathbf{x}^*_k)$ with $\mathbf{x}^*_k$ being the true k-th nearest neighbor of $\mathbf{q}$. For convenience in evaluation, we usually do not calculate the exact value of ε (cf. [18]), instead, we use recall@k as the practical measure of search accuracy, defined as

$$\text{recall@}k = \frac{|\mathcal{N}'_k(\mathbf{q}) \cap \mathcal{N}_k(\mathbf{q})|}{k}. \tag{1}$$

The performance of an ANN algorithm is usually measured by the trade-off between its search speed and accuracy. In this paper, we aim to design an ANN algorithm that achieves a superior accuracy-speed trade-off compared to existing methods.

For ease of reading, some important notations used throughout this paper are listed in Table 1.

Table 1: Important Notations

Notation	Meaning
$\mathcal{X}$	vector dataset
n, d	size, dimension of vector dataset
$\mathcal{Y}$	Transformed dataset
M	subspace number
m	index of subspace, $1 \leq m \leq M$
C^m	codebook in subspace m
$\mathbf{c}^m_j$	the j-th codeword (i.e., centroid) in subspace m
B	bits number for quantization in each subspace
K	number of codewords in each subspace, $K = 2^B$
Π	orthogonal rotation matrix
T^m	lookup table in the subspace m
$d(\mathbf{q}, \mathbf{x})$	distance between vector $\mathbf{q}$ and $\mathbf{x}$
$d_{\text{approx}}(\mathbf{q}, \mathbf{x})$	approximate distance between vector $\mathbf{q}$ and $\mathbf{x}$
I	iteration number of k-means.
σ	per-dimension standard deviation after JL transformation
ε	distance error bound (subscript indicates context)

2.2 Existing Solutions

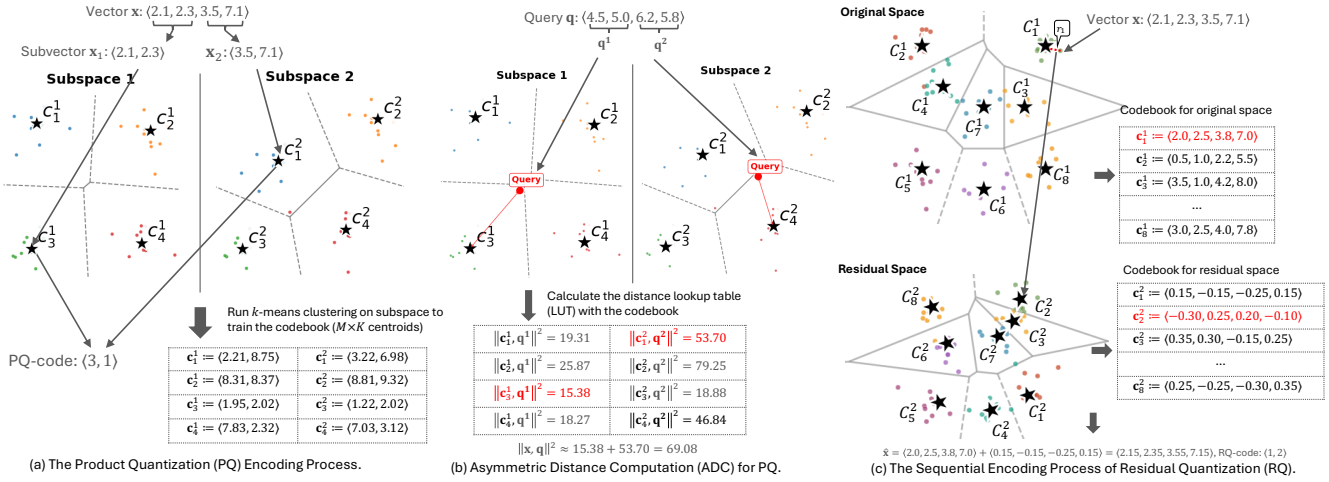
The basic idea of Vector Quantization is to turn high-dimensional vectors into short codes and construct lookup tables for fast retrieval. While a large body of literature is available on VQ (see [44] for a comprehensive survey), we restrict our presentation to the core concepts of Product Quantizers and Residual Quantizers, as they are essential for understanding the contributions of this work.

Product Quantizers (PQ). The core idea of PQ is to compress a high-dimensional vector $\mathbf{x} \in \mathbb{R}^d$ into a compact code. This is achieved by partitioning the vector into M distinct lower-dimensional sub-vectors, followed by quantizing each sub-vector into a B-bit compact code independently, and concatenating them as the final compact code. In each subspace, K centroids (i.e., codewords) are learned via k-means clustering and form a sub-codebook C^m ($1 \leq m \leq M$). As a result, each subspace is partitioned into $K = 2^B$ clusters. Then each sub-vector is mapped to the index of its nearest codeword. We denote the j-th codeword in the m-th subspace as $\mathbf{c}^m_j (1 \leq m \leq M, 1 \leq j \leq K)$ and $\mathbf{x}$'s code is denoted as $\langle j_1, j_2, \ldots, j_M \rangle$, where j_i indicates that $\mathbf{x}$'s sub-vector is mapped to $\mathbf{c}^i_{j_i}$ in the i-th subspace. As a result, K^M codewords in the full space can be enumerated and represented by storing $M \times K$ sub-codewords.

Example 1. *As illustrated in Figure 1(a), a 4-dimensional vector* $\mathbf{x} = \langle 2.1, 2.3, 3.5, 7.1 \rangle$ *is partitioned into* $M = 2$ *sub-vectors:* $\mathbf{x}^1 = \langle 2.1, 2.3 \rangle$ *and* $\mathbf{x}^2 = \langle 3.5, 7.1 \rangle$. *Each sub-vector is quantized into a 2-bit code with* $B = 2$, *hence each subspace is partitioned into* $2^B = 2^2 = 4$ *parts. The nearest codeword for* $\mathbf{x}^1$ *is the 3rd in the 1st subspace, i.e.,* $\mathbf{c}^1_3$, *and for* $\mathbf{x}^2$, *it is the 1st in the 2nd subspace, i.e.,* $\mathbf{c}^2_1$. *This yields the final PQ-code for* $\mathbf{x}$ *as the tuple of codeword IDs* $\langle 3, 1 \rangle$. *In the query stage, for a given query vector* $\mathbf{q}$, *it is first partitioned into M sub-vectors of query* $\langle \mathbf{q^1}, \mathbf{q^2}, \ldots, \mathbf{q^M} \rangle$, *then its distance lookup table (LUT) is pre-computed by storing the distances between the query's sub-vectors* $\mathbf{q^i}$ *and all corresponding sub-codewords* $\langle \mathbf{c}^i_1, \mathbf{c}^i_2, \ldots, \mathbf{c}^i_K \rangle$, $(1 \leq i \leq M)$. *Its distance to a database vector* $\mathbf{x}$ *is approximated by retrieving the corresponding pre-computed distances from the LUT with* $\mathbf{x}$*'s code* $\langle j_1, j_2, \ldots, j_M \rangle$, *i.e.,* $d(\mathbf{q}, \mathbf{x}) = \sum_{m=1}^{M} d(\mathbf{q^m}, \mathbf{c^m_{j_m}})$. *This distance computation method is also called Asymmetric Distance Computation (ADC) since it uses* $\mathbf{q}$*'s original vector while* $\mathbf{x}$*'s quantized code. It reduces the distance calculation to a small number of memory lookups and additions. For example, in Figure 1(b), the LUT of a query* $\mathbf{q} = \langle 4.5, 5.0, 6.2, 5.8 \rangle$ *is pre-computed as* $d(\mathbf{q}, \mathbf{x}) = d(\mathbf{q^1}, \mathbf{c}^1_3) + d(\mathbf{q^2}, \mathbf{c}^2_1) = 15.38 + 53.70 = 69.08$.

Residual Quantizers (RQ). RQ is designed to achieve lower quantization error than PQ by approximating a vector as the sum of several full-dimensional codewords. In stage 1, the first codebook C^1 is built to contain K centroids (i.e., codewords) $\mathbf{c}^1_j (1 \leq j \leq K)$ of the original dataset $\mathcal{X}$. For a vector $\mathbf{x}$, suppose $\tilde{\mathbf{x}}_1$ is its closest codeword in C^1. In stage 2, with the residual errors $\mathbf{r}_1 = \mathbf{x} - \tilde{\mathbf{x}}_1$, $\forall \mathbf{x} \in \mathcal{X}$ collected as a dataset $\mathcal{X}_1$, the second codebook C^2 is built to contain the K centroids of $\mathcal{X}_1$. Then $\tilde{\mathbf{x}}_2$ is selected from C^2 as the closest codeword to $\mathbf{r}_1$ and $\mathbf{r}_2 = \mathbf{r}_1 - \tilde{\mathbf{x}}_2$, $\forall \mathbf{r}_1 \in \mathcal{X}_1$ are collected as a dataset $\mathcal{X}_2$ to build the third codebook C^3 in stage 3. This continues for M stages with M codebooks constructed, and each stage of quantization aims to correct the error from the previous one. The final approximation of $\mathbf{x}$ is the sum of the codewords selected at each stage, i.e., $\hat{\mathbf{x}} = \sum_{m=1}^{M} \tilde{\mathbf{x}}_m$.

Example 2. *As illustrated in Figure 1(c), the RQ process with* $\mathbf{x} = \langle 2.1, 2.3, 3.5, 7.1 \rangle$ *and* $M = 2$. *Suppose each vector is quantized into a 3-bit compact code, then* $K = 2^3 = 8$, *hence the dataset in both the original and residual spaces is partitioned into 8 clusters. The first codeword* $\tilde{\mathbf{x}}_1 = \langle 2.0, 2.5, 3.8, 7.0 \rangle$ *is selected from* C^1 *to approximate* $\mathbf{x}$. *This leaves a residual error* $\mathbf{r}_1 = \mathbf{x} - \tilde{\mathbf{x}}_1 = \langle 0.1, -0.2, -0.3, 0.1 \rangle$,



(a) The Product Quantization (PQ) Encoding Process. (b) Asymmetric Distance Computation (ADC) for PQ. (c) The Sequential Encoding Process of Residual Quantization (RQ).

Figure 1: Core mechanisms of Product Quantization (PQ) and Residual Quantization (RQ).

based on which the second codeword $\tilde{\mathbf{x}}_2 = \langle 0.15, -0.15, -0.25, 0.15\rangle$ *is selected from* C^2. *Then the final approximation of* $\mathbf{x}$ *is* $\hat{\mathbf{x}} = \tilde{\mathbf{x}}_1 + \tilde{\mathbf{x}}_2 = \langle 2.15, 2.35, 3.55, 7.15\rangle$.

RQ's additive structure directly impacts search performance. The computational complexity for the squared Euclidean distance, $\|\mathbf{q} - \hat{\mathbf{x}}\|^2$, arises from the expansion of the vector norm $\|\hat{\mathbf{x}}\|^2$. Since the approximation $\hat{\mathbf{x}}$ is a sum of M non-orthogonal codewords, its squared norm expands to include costly cross-terms:

$$\|\hat{\mathbf{x}}\|^2 = \|\sum_{m=1}^{M} \tilde{\mathbf{x}}_m\|^2 = \sum_{m=1}^{M} \|\tilde{\mathbf{x}}_m\|^2 + \sum_{i\neq j}^{M} \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle. \tag{2}$$

The second term in the equation, representing the dot products between all pairs of codewords from different stages, must be computed. This requires $O(M^2)$ lookups, making the distance calculation more expensive than the linear $O(M)$ complexity of PQ.

3 JL-ENHANCED VECTOR QUANTIZATION

We now introduce the JL transformation along with its properties, used for our naive JL-enhanced quantization algorithm.

3.1 Theoretical Foundations of JL Transform

Our approach is built upon a fundamental preprocessing step through the orthogonal JL transformation, which is defined next.

Definition 1 (Orthogonal JL transformation). *Given a dataset* $\mathcal{X} \subset \mathbb{R}^{d\times n}$, *the orthogonal JL transformation randomizes this dataset by multiplying it by a random rotation matrix* Π, *i.e.,* $\Pi\mathcal{X} = \mathcal{Y}$, *where* $\mathcal{Y} \subset \mathbb{R}^{d\times n}$ *is the transformed dataset and* $\Pi \in \mathbb{R}^{d\times d}$ *is an orthogonal matrix.*

Given the dimension d of the dataset $\mathcal{X}$, we obtain $\Pi \in \mathbb{R}^{d\times d}$ independently of $\mathcal{X}$ by sampling a Gaussian matrix $G \in \mathbb{R}^{d\times d}$ with $G_{ij} \sim \mathcal{N}(0,1)$, computing its QR-decomposition [20] with $G = QR$, and setting $\Pi = Q$. It takes $O(d^3)$ time to obtain Π. To guarantee the ANN query accuracy, we fundamentally require that the *distance order* is preserved after transformation, i.e., for any vector $\mathbf{x} \in \mathbb{R}^d$, its kNN in $\mathcal{X}$ is the same as in $\mathcal{Y}$. Otherwise, the search implemented on $\mathcal{Y}$ will generate results that deviate from the true ones, reducing recall. The following Lemma indicates that the *distance order* can be well preserved after the JL transformation.

Example 3. *Consider a 4-dimensional dataset. We generate the orthogonal matrix* Π *as follows:*

(1) Generate a random matrix $\mathbf{G} \in \mathbb{R}^{4\times 4}$ *with entries* $g_{ij} \sim \mathcal{N}(0,1)$:

$$\mathbf{G} = \begin{pmatrix} 1.24 & -0.51 & 0.83 & -0.27 \\ 0.62 & 1.17 & -0.45 & 0.91 \\ -0.38 & 0.74 & 1.32 & -0.56 \\ 0.89 & -0.23 & 0.67 & 1.08 \end{pmatrix}$$

(2) Apply QR decomposition on G *with* $\mathbf{G} = \mathbf{QR}$, *and take* $\Pi = Q$:

$$\Pi = \mathbf{Q} \approx \begin{pmatrix} 0.73 & -0.23 & 0.27 & 0.58 \\ 0.37 & 0.85 & -0.37 & 0.05 \\ -0.22 & 0.47 & 0.85 & 0.07 \\ 0.53 & -0.07 & 0.24 & -0.81 \end{pmatrix}$$

(3) For a vector $\mathbf{x} = [0.71, 0.36, 0.21, 0.57]^T$, *the transformation yields (values rounded to 2 decimal places):*

$$\mathbf{y} = \Pi\mathbf{x} \approx [0.82, 0.51, 0.23, -0.06]^T$$

Lemma 1 (Distance Preservation). *For any vectors* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, *the transformation* Π *preserves their distance, i.e.,* $d(\mathbf{x}, \mathbf{y}) = d(\Pi\mathbf{x}, \Pi\mathbf{y})$, *when* $d(,)$ *is inner product, cosine, or Euclidean distance.*

We note that these three distances are typically used in ANN benchmarks. Since the distance is preserved after the transformation, it is easy to see that the *distance order* is preserved as well, which lays a theoretically solid foundation for further design.

Next, we use Lemma 2 (which is derived from [70]) to demonstrate that the transformed dataset $\mathcal{Y}$ has a determined distribution [70], though the distribution of the original dataset $\mathcal{X}$ is unknown.

Lemma 2. *Let* $\mathbf{x} \in \mathbb{R}^d$ *be a* d*-dimensional vector, where* $r = \|\mathbf{x}\|$ *is the length of* $\mathbf{x}$. $\Pi \in \mathbb{R}^{d\times d}$ *is a random orthogonal matrix; we denote* $\mathbf{y} = \Pi\mathbf{x}$, *where each dimension* y_i *has a mean of* 0 *and a*

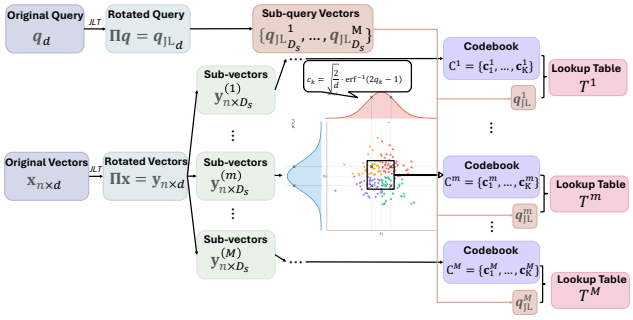


Figure 2: The workflow of JQ

variance of r^2/d. For $\mathbf{x} \in \mathcal{X}$, we define the average per-dimension variance as $\sigma^2 = \mathbb{E}_{\mathbf{x}}[\|\mathbf{x}\|^2]/d$, where $\mathbb{E}_{\mathbf{x}}[\cdot]$ denotes expectation over $\mathcal{X}$. As the dimension d grows, the distribution of each dimension y_i can be approximated as a one-dimensional normal distribution $y_i \sim \mathcal{N}(0, \sigma^2)$.

From the above Lemma, we can deduce that each dimension in the transformed dataset $\mathcal{Y}$ follows a near-Gaussian distribution. This indicates that we can use and share one scalar codebook among all dimensions for vector quantization. Meanwhile, with the determined distribution, we can generate the codebook quickly by eliminating the expensive iterations in k-means clustering. Specifically, for one codebook of size $K = 2^B$, where B denotes the number of bits per dimension, we can compute each Lloyd-Max codeword in $O(1)$ time by using the inverse cumulative distribution function:

$$c_i = \sigma\sqrt{2} \cdot \mathrm{erf}^{-1}(2q_i - 1), \quad q_i = \frac{i - 0.5}{K}, \quad i = 1, \ldots, K \tag{3}$$

where $\sigma^2 = \mathbb{E}[\|\mathbf{x}\|^2]/d$, erf^{-1} is the inverse of the error function $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t$. This reduces the time complexity of codebook generation from $O(I \cdot n \cdot K)$ with k-means clustering to $O(K)$, where I is the number of k-means iterations.

Importantly, this scalar codebook is of high precision as each dimension is quantized independently, but can result in slow query processing. For each query, it takes $O(dK)$ time to build the lookup tables, since each dimension has one lookup table with K distance calculations. The *Asymmetric Distance Computation* for each vector takes $O(d)$ time. Then it takes $O(d(K+n))$ time in total to compute the distance to n vectors in the dataset to obtain the query result, which is even slower than the brute-force search with $O(dn)$ time. Therefore, the query processing could be rather slow.

3.2 JL-Enhanced Quantization (JQ)

To address the slow query answering with scalar codebooks, we introduce a novel algorithm, JL-enhanced Quantization (JQ). As illustrated in Figure 2, we first apply the JL transformation to the dataset and partition the transformed dataset into M subspaces. Next, in each subspace, one codebook is constructed through the Cartesian product of each dimension's Lloyd-Max codewords, which is generated by the inverse cumulative distribution function under the Gaussian distribution. Given a query vector, we transform it and partition it into M query sub-vectors and their distances to codewords in the corresponding codebook are calculated to form the lookup table. Then the distance from the query vector to database vectors can be computed with *Asymmetric Distance Computation* by checking the lookup tables, to obtain the final nearest neighbors. Specifically, JQ consists of three phases: (1) JL transformation and space partitioning, (2) codebook construction, and (3) query processing, which are elaborated below.

JL Transformation and Space Partitioning. We apply the orthogonal JL transformation to $\mathcal{X}$ and compute $\mathbf{y} = \Pi\mathbf{x}$ for each data vector $\mathbf{x} \in \mathcal{X}$. The transformed vector $\mathbf{y}$ is then partitioned into subspaces $\mathbf{y} = [\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(M)}]$ where $\mathbf{y}^{(m)} \in \mathbb{R}^{D_s} (1 \le m \le M)$ and $D_s = d/M$ is the dimension of each subspace. For the m-th subspace, $\mathbf{y}^{(m)}$ is quantized to the nearest codeword $\hat{\mathbf{y}}^{(m)} = \arg\min_{\mathbf{c} \in C^{(m)}} \|\mathbf{y}^{(m)} - \mathbf{c}\|^2$, where $C^{(m)} = \{\mathbf{c}_k^m\}_{k=1}^K$ is the codebook for subspace m with $K = 2^B$ codewords. The final code of each vector is the sequence of subspace indices: $\mathrm{code}(\mathbf{x}) = [q_1, q_2, \ldots, q_M]$, where q_m is the index of $\hat{\mathbf{y}}^{(m)}$ in $C^{(m)}$. It takes $O(nd^2)$ time to implement the JL transformation on $\mathcal{X}$.

Codebook Construction. The codebook is constructed by selecting centroids, i.e., codewords, in each subspace of the transformed dataset $\mathcal{Y}$. Based on the near-Gaussian distribution of $\mathcal{Y}$, we propose to calculate codewords directly without leveraging the k-means method. First, we show that the centroids of two similar distributions are close enough, based on the following Lemma.

Lemma 3. *Let P and Q be two probability distributions on $\mathbb{R}^{D_s}$ with finite variance. If the distributions are close in the sense that their 2-Wasserstein distance [66] satisfies $W_2(P, Q) \le \varepsilon$, then their optimal centroids under squared error loss satisfy $\|\mathbf{c}_P - \mathbf{c}_Q\| \le \varepsilon$.*

We construct the codebook of each subspace by taking the Cartesian Product of Lloyd-Max codewords across D_s dimensions. To get a codebook of size $K = 2^B$, we need to compute K^{1/D_s} Lloyd-Max codewords to obtain $K = (K^{1/D_s})^{D_s}$ centroids with Cartesian Product. Each codeword can be calculated with $O(1)$ using:

$$c_{ji}^m = \sqrt{\frac{2}{d}} \cdot \mathrm{erf}^{-1}(2q_i - 1), \quad q_i = \frac{i - 0.5}{K^{1/D_s}} \tag{4}$$

where $i = 1, \ldots, K^{1/D_s}, j = 1, \ldots, D_s$. Then the codebook of the subspace m is constructed through Cartesian Product of Lloyd-Max codewords in each dimension j: $C^m = \mathbf{c}^{m1} \times \mathbf{c}^{m2} \times \cdots \times \mathbf{c}^{mD_s}$ with $\mathbf{c}^{mj} = [c_{j1}^m, c_{j2}^m, \ldots, c_{jK^{1/D_s}}^m]^T$ and $|C^m| = K$. Therefore, the time complexity to construct the codebooks is $O(MK)$ in total, which is much smaller than the construction by k-means with $O(InKD_sM)$.

Query Processing. JQ enables efficient distance computation by checking the lookup tables. In the query processing stage, given a query vector $\mathbf{q}$, we start by applying the JL transformation on it with $\mathbf{q}_{\mathrm{JL}} = \Pi\mathbf{q}$ and divide it into M sub-query vectors $\{\mathbf{q}_{\mathrm{JL}}^m\}, (1 \le m \le M)$. Then $\mathbf{q}$'s lookup tables T^m in each subspace m are constructed with $T^m[k] = d(\mathbf{q}_{\mathrm{JL}}^m, \mathbf{c}_k^m)$ $(1 \le k \le K$ and $1 \le m \le M)$, where $\mathbf{c}_k^m$ denotes the k-th codeword in C^m. Its distance to a database vector $\mathbf{x}$ is $d(\mathbf{q}, \mathbf{x}) = d(\mathbf{q}_{\mathrm{JL}}, \Pi\mathbf{x})$ according to Lemma 1. Denoting $\Pi\mathbf{x} = \mathbf{y}$, then $d(\mathbf{q}, \mathbf{x}) = d(\mathbf{q}_{\mathrm{JL}}, \mathbf{y}) = \sum_{m=1}^M d(\mathbf{q}_{\mathrm{JL}}^m, \mathbf{y}^{(m)})$. Suppose $\mathbf{x}$'s quantization code is $[q_1, \ldots, q_M]$, where q_m is the index of the closest codeword to $\mathbf{q}_{\mathrm{JL}}^m$ in C^m. The distance can be approximated with *Asymmetric Distance Computation* as $d_{\mathrm{approx}}(\mathbf{q}, \mathbf{x}) = \sum_{m=1}^M d(\mathbf{q}_{\mathrm{JL}}^m, \hat{\mathbf{y}}^{(m)}) = \sum_{m=1}^M T^m[q_m]$, requiring M lookups with $O(M)$ time.

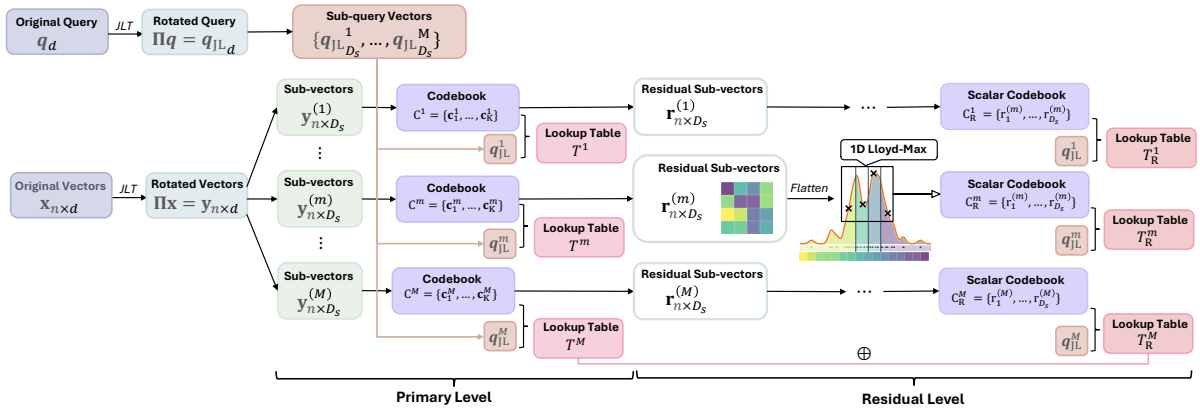


Figure 3: The architecture of JHQ

Therefore, it takes $O(Mn + dK + n\log k)$ time in total for processing each query, where $O(Mn)$ is the time to compute the distance from $\mathbf{q}$ to n vectors, $O(MKD_s) = O(dK)$ is the time to build the lookup table and $O(n\log k)$ is the time to sort distances for the final kNN results. It is faster than the scalar codebook (cf. Section 3.1) and is the same as that of PQ. Nevertheless, with the independence among dimensions in the transformed dataset based on Lemma 2, our algorithm will achieve higher accuracy, since PQ's performance has been constrained by the subspace dependence (cf. Section 1).

3.3 Distance Error Bound

We now show that the distance error in the JQ algorithm can be bounded under the Euclidean distance metric.

Theorem 4. *Given a query vector* $\mathbf{q}$ *and a dataset vector* $\mathbf{x} \in \mathcal{X}$, *the Euclidean distance error can be bounded with probability* P_ε *as follows:* $\varepsilon_{\mathcal{JQ}} = d(\mathbf{q},\mathbf{x}) - d_{approx}(\mathbf{q},\mathbf{x}) \le \sqrt{MD_s}\cdot(-c_1)$, *where* $P_\varepsilon = \Phi\left(\frac{-c_1}{\sigma}\right) - \Phi\left(\frac{c_1}{\sigma}\right)$ *with* $\sigma = \sqrt{\mathbb{E}[\|\mathbf{x}\|^2]/d}$, $c_1 = \sigma\sqrt{2}\cdot erf^{-1}\left(\frac{1}{K^{1/D_s}} - 1\right), (c_1 < 0)$, *and* Φ *is the standard normal CDF.*

Although the distance error of JQ can be bounded, it faces a scalability issue. With the dataset partitioned into M subspaces and B bits for a quantization in each subspace, there are 2^{MB} quantization codes in total. For a dataset of size n, the average collision count of quantization code is $n/2^{MB}$. As n becomes larger, it is more likely that two vectors will share one quantization code. In other words, it is more difficult to discriminate between vectors based on their quantization code which, in turn, could directly affect the query accuracy. As a result, we need to quantize vectors with a longer code to better differentiate them, such that the query accuracy can be maintained. However, a longer code will require larger index storage and reduce the speed of query processing.

4 JL-ENHANCED HIERARCHICAL QUANTIZATION (JHQ)

To resolve the scalability issue, i.e., the conflict between query speed and accuracy on larger datasets, we introduce a novel quantization structure that effectively balances this trade-off.

4.1 Hierarchical Quantization Architecture

The general idea of JHQ is to quickly identify the candidates with short codes and then refine the quantization with longer codes for higher accuracy. As illustrated in Figure 3, our JHQ algorithm uses a two-level architecture: in the primary level, the transformed dataset is partitioned into M subspaces and a codebook is constructed in each subspace. In the residual level, we use the database sub-vector's closest codeword in the codebook to approximate it and denote the difference as the residual sub-vector. Then every one-dimensional value from the residual sub-vectors is collected and one-dimensional Lloyd–Max quantization is implemented to obtain their centroids, which form the scalar codebook. Given a query vector, we transform it and partition it into M query sub-vectors. In each subspace, lookup tables are constructed by calculating the distance from query sub-vector to each codeword in the codebook; the scalar lookup table is built by calculating the difference between the values in query sub-vector and each codeword in scalar codebook. Then the composite distance is calculated through ADC by checking lookup tables in both primary and residual levels to get the final query results. By applying the expensive and high-precision quantization only on promising candidates, this design enables high query accuracy without compromising query speed.

Next, we discuss the principle of index (i.e., codebook) construction (Section 4.2) and the query processing (Section 4.3).

4.2 JHQ Codebook Construction

Codebooks for the primary and residual levels are built separately.

Primary Level. The primary level implements JQ as described in Section 3. Each transformed vector is partitioned into M subspaces of dimension $D_s = d/M$, and each subspace is quantized with B bits. In each subspace, the codebook contains $K = 2^B$ centroids, calculated by the Cartesian product of the Lloyd–Max codewords in each dimension. This creates compact codes that enable fast Asymmetric Distance Computation with lookup tables of size $M\times 2^B$ for quick query answering. For a transformed vector $\mathbf{y} = \Pi\mathbf{x}$, each sub-vector $\mathbf{y}^{(m)}$ is quantized to its nearest codeword $\hat{\mathbf{y}}^{(m)}$ in C^m. This generates the residual, i.e., the error $\mathbf{r}^{(m)} = \mathbf{y}^{(m)} - \hat{\mathbf{y}}^{(m)}$.

Residual Level. Traditional residual quantization methods learn full-dimensional vector codebooks for successive residual approximations, leading to exponential training complexity and quadratic distance computation due to cross-term interactions. We introduce a novel scalar decomposition approach to achieve linear complexity while maintaining superior quantization accuracy.

The key insight of our scalar approach lies in exploiting the dimensional independence established by the JL transformation in Lemma 2. Based on that, we can deduce the dimensional independence of the quantization residual as follows.

Lemma 5. *Given a transformed vector* $\mathbf{y} = \Pi\mathbf{x}$*, its sub-vectors* $\{\mathbf{y}^{(m)}\}$ $(1 \le m \le M)$ *in each subspace, and its closest codewords* $\{\hat{\mathbf{y}}^{(m)}\}$*, then the quantization residual is* $\mathbf{R} = \text{concat}_{m=1}^{M}\, \mathbf{r}^{(m)}$ *with* $\mathbf{r}^{(m)} = \mathbf{y}^{(m)} - \hat{\mathbf{y}}^{(m)}$, $\mathbf{r}^{(m)} \in \mathbb{R}^{D_s}$*, where each dimension of* $\mathbf{R}$ *is mutually independent.*

Next, we start to construct the scalar codebook in residual level. Since each dimension of the quantization residual is independent based on Lemma 5, for each dimension in subspace m, we build and share one scalar codebook C_R^m. First, we collect every one-dimensional value from the quantization residual $\mathbf{r}^{(m)} = \mathbf{y}^{(m)} - \hat{\mathbf{y}}^{(m)}, \forall \mathbf{y} \in \mathcal{Y}$ and form a one-dimensional dataset R^m:

$$R^m = \{r_j^{(m)}\}, 1 \le j \le D_s, r_j^{(m)} = y_j^{(m)} - \hat{y}_j^{(m)}, \mathbf{y} \in \mathcal{Y} \tag{5}$$

where $y_j^{(m)}, \hat{y}_j^{(m)}$ are the values of the j-th dimension of $\mathbf{y}^{(m)}, \hat{\mathbf{y}}^{(m)}$, respectively. Then we apply one-dimensional k-means clustering on R^m to get $K_r = 2^{B_r}$ centroids as the codewords to form C_R^m, where B_r is the number of bits for quantization in each dimension of residual level. This takes $O(n \cdot K_r)$ time.

To quantize vectors, we use codebooks in both the primary and residual levels. For each sub-vector $\mathbf{y}^{(m)}$ of the transformed vector $\mathbf{y}$, its quantization includes two parts: 1) its closest centroid $\hat{\mathbf{y}}^{(m)}$ in C^m; and 2) the concatenation of its quantization residual's closest centroids $\hat{r}_j^{(m)}$ in C_R^m for each dimension $j (1 \le j \le D_s)$. As a result, each vector in the dataset is quantized as $[q_1, q_2, \ldots, q_M]$ in the primary level and $\{[q_{m1}^r, \ldots, q_{mD_s}^r]\}$ in the residual level, where q_m is the index of $\hat{\mathbf{y}}^{(m)}$ in C^m and q_{mj}^r is the index of $\hat{r}_j^{(m)}$ in C_R^m. It takes $O(MK_r)$ space to store the codebooks of the residual level.

4.3 JHQ Query Processing

The hierarchical structure enables efficient distance computation through two-level lookup tables to improve the query accuracy. We leverage the ADC to calculate distance in both the primary and residual levels, without incurring expensive quadratic cross-term computation in the existing additive quantization.

Given a query vector $\mathbf{q}$, we apply the JL transformation to it and its distance to a database vector $\mathbf{x}$ is $d(\mathbf{q}, \mathbf{x}) = d(\mathbf{q}_{\text{JL}}, \mathbf{y})$ with $\mathbf{q}_{\text{JL}} = \Pi\mathbf{q}$ and $\mathbf{y} = \Pi\mathbf{x}$ according to Lemma 1. Next, the query processing includes two components: building the lookup tables and leveraging them for ADC distance computation.

Build lookup tables. The primary level's lookup tables $\{T^m\}$ are constructed in the same way as $\mathcal{JQ}$ by referring to Section 3.2. In the residual level, there is one lookup table T_R^{mj} for each dimension j in the subspace m. It is constructed by calculating the distance between $\mathbf{q}_{\text{JL}}$ and codewords in the codebook C_R^m in each dimension with $T_R^{mj}[k] = d(\mathbf{q}_{\text{JL}}^{mj}, c_{Rk}^m), (1 \le k \le 2^{B_r})$, where $\mathbf{q}_{\text{JL}}^{mj}$ is $\mathbf{q}_{\text{JL}}$'s j-th value in subspace m and c_{Rk}^m is the k-th codeword in C_R^m. Thus, we have $M \times D_s = d$ lookup tables in total in the residual level, and it takes $O(d\, 2^{B_r}) = O(dK_r)$ to construct them.

ADC distance computation. Based on vector quantization, the distance is $d(\mathbf{q}_{\text{JL}}, \mathbf{y}) = d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}} + \hat{\mathbf{r}})$, whose calculation is inefficient requiring a much larger codebook. To facilitate faster query processing, we speed up this computation by calculating $d(\mathbf{q}_{\text{JL}}, \mathbf{y})$ as $d(\mathbf{q}_{\text{JL}}, \mathbf{y}) = d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}}) + d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{r}})$, where $\hat{\mathbf{y}}$ is the quantized vector of $\mathbf{y}$ by leveraging the codebook C^m in the primary level; $\hat{\mathbf{r}}$ is the quantized vector of the quantization error $\mathbf{r} = \mathbf{y} - \hat{\mathbf{y}}$ by using C_R^m.

Theorem 6. *For the inner product, squared Euclidean, and cosine distance metrics, the distance from* $\mathbf{q}_{\text{JL}}$ *to the reconstruction can be decomposed as follows: for inner product and cosine distance,* $d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}}+\hat{\mathbf{r}}) = d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}})+d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{r}})$*, and for squared Euclidean distance,* $d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}} + \hat{\mathbf{r}}) = d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}}) + d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{r}}) - \|\mathbf{q}_{\text{JL}}\|^2 + 2\langle\hat{\mathbf{y}}, \hat{\mathbf{r}}\rangle$.

Theorem 6 shows that this distance computation does not degrade query accuracy as it preserves the exact distances for inner product and cosine, and recovers the squared Euclidean distance with a simple correction term. As both the query and dataset are partitioned into M subspaces, we decompose the distance measure in the full dimensional space into a summation of distances within subspaces. Specifically, the distance in the primary level is

$$d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}}) = \sum_{m=1}^{M} d(\mathbf{q}_{\text{JL}}^m, \hat{\mathbf{y}}^{(m)}) \tag{6}$$

Since each dimension of the transformed dataset is independent based on Lemma 2, it is easy to see that each subspace is also independent. So we can guarantee that the cross-subspace distance terms vanish, making the additive distance equivalent to the full-dimensional distance. With $\mathbf{x}$'s quantization code $[q_1, q_2, \ldots, q_M]$ in the primary level, the primary distance can finally be calculated by checking the lookup tables $\{T^m\}$ as $d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}}) = \sum_{m=1}^{M} T^m[q_m]$. Then the distance in the residual level is calculated as

$$d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{r}}) = \sum_{m=1}^{M} \sum_{j=1}^{D_s} d(q_j^m, \hat{r}_j^{(m)}) \tag{7}$$

Where q_j^m denotes the j-th coordinate of $\mathbf{q}_{\text{JL}}^m$. Similar to the primary level, the dimensional independence can ensure that the additive distance among each dimension is equal to the full-dimensional distance in the residual level. With the $\mathbf{x}$'s quantization code $\{[q_{m1}^r, \ldots, q_{mD_s}^r]\}$ $(1 \le m \le M)$ in the residual level, the residual distance can finally be calculated by checking the lookup tables

$$d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{r}}) = \sum_{m=1}^{M} \sum_{j=1}^{D_s} T_R^{mj}[q_{mj}^r] \tag{8}$$

So distance $d(\mathbf{q}, \mathbf{x})$ is approximated with quantization as

$$d_{\text{approx}}(\mathbf{q}, \mathbf{x}) = \sum_{m=1}^{M} T^m[q_m] + \sum_{m=1}^{M} \sum_{j=1}^{D_s} T_R^{mj}[q_{mj}^r] \tag{9}$$

As a result, it takes $O(M + d)$ time to compute the distance per vector , which is much faster than traditional residual quantization methods (e.g., IRVQ, LSQ++) with $O(M^2)$ time due to cross-term computations. So it takes $O(n(M+d+\log k)+d(K+K_r))$ to process

each query, where $O(n(M+d))$ is the time to compute the distance from $\mathbf{q}$ to n vectors, $O(n \log k)$ is the time to sort distances to get the kNN results, $O(dK)$ and $O(dK_r)$ are the times to build the lookup tables in the primary level and residual level, respectively. However, query processing is largely slowed down by the residual level. Since the primary level could achieve high accuracy with the distance error bounded, when we find results among n vectors for top-k nearest neighbors, it is not necessary to compute the composite distance for higher accuracy at the cost of much slower query speed.

To better balance the accuracy–speed trade-off, we propose a *selectively refine strategy* to subtly leverage the residual distance without imposing too much burden on query processing. Instead of starting with calculating the composite distance to each vector all at once, we first quickly calculate the primary distance to all n vectors to get a smaller set of candidates $\mathcal{Z}$ ($|\mathcal{Z}| = \alpha k \ll n, \alpha > 1$) with $O(Mn + n\log(\alpha k))$ time. Then the residual distance is computed only on $\mathcal{Z}$ with $O(\alpha k d)$ time. Along with the primary distance on $\mathcal{Z}$, the composite distance on $\mathcal{Z}$ can be obtained and used to select the query results more accurately with $O(\alpha k \log k)$ time. Finally, it takes $O(n(M + \log(\alpha k)) + \alpha k(d + \log k))$ time to search the query results, which is faster than applying the composite distance on all n vectors. The parameter α provides a tunable and flexible trade-off between search accuracy and speed: higher values enlarge the candidate pool to improve recall but increase distance computation cost, while lower values prioritize speed but risk missing neighbors. The pseudocode of JHQ's query processing is shown in Algorithm 1.

Algorithm 1: JHQ-Query($\mathbf{q}, \mathcal{Y}$)

Data: query vector $\mathbf{q}$, transformed dataset $\mathcal{Y}$
Result: approximate top-k nearest neighbors of $\mathbf{q}$

$\mathbf{q}_{\text{JL}} = \Pi\mathbf{q}$;
forall $\mathbf{y} \in \mathcal{Y}$ **do**
  Calculate the primary distance $d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{y}})$
Select the top-αk nearest neighbors $\mathcal{Z}$ from $\mathcal{Y}$ based on primary distance;
forall $\mathbf{z} \in \mathcal{Z}$ **do**
  Calculate the residual distance $d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{r}})$;
  Get the composite distance $d(\mathbf{q}_{\text{JL}}, \mathbf{z}) = d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{z}}) + d(\mathbf{q}_{\text{JL}}, \hat{\mathbf{r}})$;
Select the top-k nearest neighbors $\mathcal{R}_k$ from $\mathcal{Z}$ based on composite distance;
return $\mathcal{R}_k$;

4.4 Distance Error and Index Discussion

We establish that JHQ achieves a provably tighter worst-case error bound than JQ by leveraging hierarchical quantization.

Theorem 7. *JHQ achieves a distance error bound* $\varepsilon_{\text{JHQ}} \leq \sqrt{d}\,\Delta_{\max} = \frac{r_1 \varepsilon_{\text{JQ}}}{(-c_1)2^{B_r}}$, *where* $\Delta_{\max}$ *is the worst-case quantization error per dimension,* ε_{JQ} *is JQ's error bound from Theorem 4,* c_1 *and* c_2 *are the two leftmost adjacent centroids defined via* $c_i = \sigma\sqrt{2} \cdot erf^{-1}(2q_i - 1)$ *in Equation* (4) *and* $r_1 = |c_1 - c_2|/2$.

When it comes to indexing, in particular integration with Inverted File Index (IVF), we note that the traditional IVF-based VQ methods implement the IVF first, get each vector's residual by subtracting its closest centroid from the vector, and apply quantization on those residuals. As a result, their quantization quality will be largely affected by the centroids in the IVF. Unlike them, we only use IVF to partition the dataset into clusters and index the vectors with the cluster ID, to roughly prune the candidates and narrow down the search space. So the quantization quality of our algorithm is not affected by the IVF.

Regarding updates, our framework naturally supports them: *(i) Insertion:* when a new vector $\mathbf{x}$ is inserted into $\mathcal{X}$, the codebook remains unchanged. To support accurate query processing, we calculate the quantized code of $\mathbf{x}$ (based on Section 3.2) and add it to the index. *(ii) Deletion:* when an existing vector $\mathbf{x} \in \mathcal{X}$ is deleted, the codebook remains unchanged. We mark $\mathbf{x}$ as deleted such that it will not be visited or included in the query results.

5 EXPERIMENTS

Experimental Platform. All experiments are conducted on a server equipped with an AMD EPYC 9654 @ 3.7GHz CPU (96 cores/192 threads), and 576GB DDR5 RAM @ 4800MHz. We compile all methods with g++ 14.2.0 in release mode, with default compilation flags and optimization level `-O3` under Ubuntu 24.04. Index construction was parallelized over 32 threads, and search performance was measured in a single-thread setting.

Baseline Methods. We compare JQ and JHQ, against state-of-the-art quantization methods from each major category as discussed in Section 6: PQ [33] and OPQ [24] for PQ-based methods; IRVQ [38] for RVQ-based methods; LSQ++ [43] for AQ-based methods; and Extended-RaBitQ [21, 22]. We include the `Brute-force` exact nearest neighbor search as an accuracy upper bound, which performs linear scan over all database vectors computing exact ℓ_2 distances without any approximation or indexing. For each baseline method, we use efficient publicly available C++ implementations from FAISS [15] for fair comparison.

Datasets. We evaluate on six public vector datasets covering image and text embeddings: (i) *OpenAI3-1536* [51] and (ii) *OpenAI3-3072* [52] each contains 1M `text-embedding-3-large` [49] embeddings generated for the first 1M entries of the BEIR benchmark [61]; (iii) *Vogue-768* [6] contains 933K `vit-base-patch16-224` [27] ViT [69] embeddings of Vogue Runway images; (iv) *arXiv-Abstracts-768* [62] comprises 2.3M `instructor-xl` [57] embeddings of all arXiv paper abstracts as of May 18, 2023; (v) *Bge-M3-1024* [65] is the Italian subset of the `BGE-M3` [12] Wikipedia-2024-06 dataset (10M vectors); (vi) *Stella-TREC24-1024* [63] contains 17.8M titles and abstracts from TREC24 BioGen, which are encoded using the `stella_en_1.5B_v5` embedding model [71]. For queries, we uniformly sample and remove 1,000 vectors from the datasets as queries. Table 2 summarizes the dataset characteristics, including the database size (N), dimensionality (D), local intrinsic dimensionality (LID [2]), and relative contrast (RC [29]). LID reflects the local complexity or manifold dimensionality of a dataset, with higher LID values implying harder dataset. RC is the ratio between the average pairwise distance and the nearest neighbor distance, with smaller RC values indicating more challenging datasets. Although our algorithms are designed to solve the scalability issue, they are essentially in-memory algorithms. Our largest dataset *Stella-TREC24-1024* now contains 17,776,615 vectors with 1,024 dimensions, whose size almost reaches our memory bottleneck. Therefore, we do not include

larger datasets in our experiments, and investigation of on-disk indexing is left for future work.

Parameter Settings. For fair comparison across all methods, we systematically tune parameters to achieve optimal accuracy-efficiency trade-offs on each dataset. For PQ, OPQ, JQ, and JHQ, we sweep the number of subspaces M to keep the subspace dimension $D_s = d/M$ around 8, following the common practice [15]. We set the value of M as: $M \in \{48, 96, 192, 384\}$ for 768-d, $M \in \{64, 128, 256\}$ for 1024-d, $M \in \{32, 64, 128, 192, 256\}$ for the 1536-d, and $M \in \{32, 64, 128, 192, 256, 384\}$ for 3072-d dataset. For IRVQ and LSQ++, we use the FAISS implementation employing product quantization to improve indexing efficiency, following the approach in [46]. We enable this feature and set subspace counts as $M \in \{48, 96\}$ for 768-d datasets, $M \in \{64, 128, 256\}$ for 1024-d datasets, $M \in \{32, 64\}$ for 1536-d dataset, and $M \in \{32, 64, 128, 192\}$ for 3072-d dataset. We set $B = 8$ bits per subspace for PQ, OPQ, IRVQ, and LSQ++, and thus $K = 2^B = 256$, following common practice in [15].

For baseline methods' unique parameters, we use their default values recommended in FAISS [15]: OPQ employs 50 outer iterations with 4 PQ training iterations per round (40 for the first round) and limits training to 65536 points via resampling; LSQ++ uses 25 training iterations with 16 encoding ILS iterations and 8 ICM iterations; IRVQ applies uniform bit allocation across subquantizers with decompression-based search. For PQ, OPQ, IRVQ, and LSQ++, which require codebook training, their indices are trained on sampled datasets of 100K vectors following standard practice in FAISS [15]. For Extended-RaBitQ, we test $B = 8$ and $B = 1$ variants.

For our proposed methods, we evaluate JQ with $B = 8$ bits per subspace using the same M values as other baselines, and JHQ with primary–residual configurations $[B, B_r] \in \{[8, 4], [4, 8], [8, 8]\}$ and selective refinement factor $\alpha \in \{2.0, 4.0, 8.0\}$.

Experimental Configuration. To match real-world deployment scenarios, we exclude reranking from our evaluation to ensure fair comparison of quantization-method performance. Reranking would compromise our evaluation objectives for several reasons: first, it degrades query performance when we aim to measure optimal query efficiency; second, reranking requires either loading the entire dataset into memory to compute exact ℓ_2 distances or performing on-demand vector loading when the dataset exceeds memory capacity. The former violates the core memory-saving principle of quantization and introduces prohibitive computational overhead, while the latter becomes impractical for high-dimensional large-scale datasets and will severely increase query latency. In addition, we build IVF for all methods following FAISS [15] by setting the number of clusters to 4,096, and the number of probed clusters for query processing, and set nprobe $\in \{1, 2, 4, 8, 16, 32, 64, 128\}$.

Evaluation Metrics. We evaluate search performance using the speed-accuracy trade-off, where accuracy is measured as recall@10 (defined in Equation 1) and speed is measured in queries per second (QPS). We also evaluate index construction time, setting a time limit of 120,000 seconds (33 hours 20 minutes) for each parameter configuration of each algorithm. Index construction time exceeding this limit is excluded from evaluation.

Table 2: Statistics of the evaluation datasets.

Dataset	N	D	LID	RC
OpenAI3-1536	999,000	1,536	28.1	1.29
OpenAI3-3072	999,000	3,072	28.4	1.27
Vogue-768	932,328	768	32.5	1.54
arXiv-Abstracts-768	2,253,198	768	31.8	1.50
BGE-M3-1024	10,091,524	1,024	30.8	1.39
Stella-TREC24-1024	17,776,615	1,024	20.1	1.65

5.1 Speed-Accuracy Trade-off

We compare the speedaccuracy trade-off of our methods with six baselines and a brute-force search on three representative datasets (OpenAI3-3072, arXiv-Abstracts-768, and Stella-TREC24-1024), as shown in Figure 4a. Results on the remaining three datasets, which support similar conclusions, are reported in Appendix B.1.

Our JQ and JHQ achieve the best speed-accuracy trade-offs compared to baseline methods on most datasets. They are much faster than IRVQ and LSQ++, since the latter need expensive cross-term distance computations between non-orthogonal codewords with $O(M^2)$ time for one distance computation during query processing, while JQ only takes $O(M)$ and JHQ takes $O(M+d)$ time. Meanwhile, JQ is more accurate than PQ and OPQ under the same throughput. PQ suffers accuracy loss when the independence assumption fails and OPQ attempts to relax the assumption but with limited improvement. In contrast, our JQ removes the assumption by leveraging the dimensional independence with the JL transformation. As a result, the near-optimal codewords can be obtained in JQ as proved in Lemma 3, which contributes to high query accuracy. For example, with QPS=1000 and $M = 192$ in the arXiv-Abstracts-768 dataset, the recall of PQ, OPQ and JQ is 57%, 73% and 89%, respectively. Besides, our JQ performs better than RaBitQ 1-bit and 8-bit, because their geometry projection requires quantizing each dimension separately, to achieve theoretical error bound. As a result, the quantized code of RaBitQ 1-bit is short, which leads to fast query but with low accuracy ($\lesssim$ 70% on most datasets); that of RaBitQ 8-bit is long, which achieves high accuracy comparable to other methods but incurs slow query processing.

Dataset Characteristics. The dataset characteristics also affect the performance. When the dataset size N grows, JHQ shows a better speedaccuracy trade-off. On million-scale datasets, JQ already achieves high accuracy with short codes, so the extra computation in the residual level of JHQ brings limited gains but slows queries. For example, on OpenAI3-3072 at 90% recall, JQ uses 1024-bit codes with 1075 QPS, while JHQ uses 128-bit codes in the primary level with 154 QPS. When datasets increase to ten-million-scale, queries are slower since the time spent in distance calculation is proportional to the dataset size. Then JHQ can achieve more time gain for queries with shorter code in the primary level, which offsets the overhead in the residual level. For instance, on Stella-TREC24-1024 dataset with 93% recall, JQ uses 1024-bit codes with 391 QPS, while JHQ uses 512-bit codes in the primary level with 889 QPS. Other characteristics, i.e., dimensionality, LID and RC, could also affect the performance of an ANN algorithm, since datasets become more challenging with an intensified curse of dimensionality with higher dimensionality, higher LID and smaller RC. Nevertheless, our algorithms consistently outperform the baselines across all datasets,

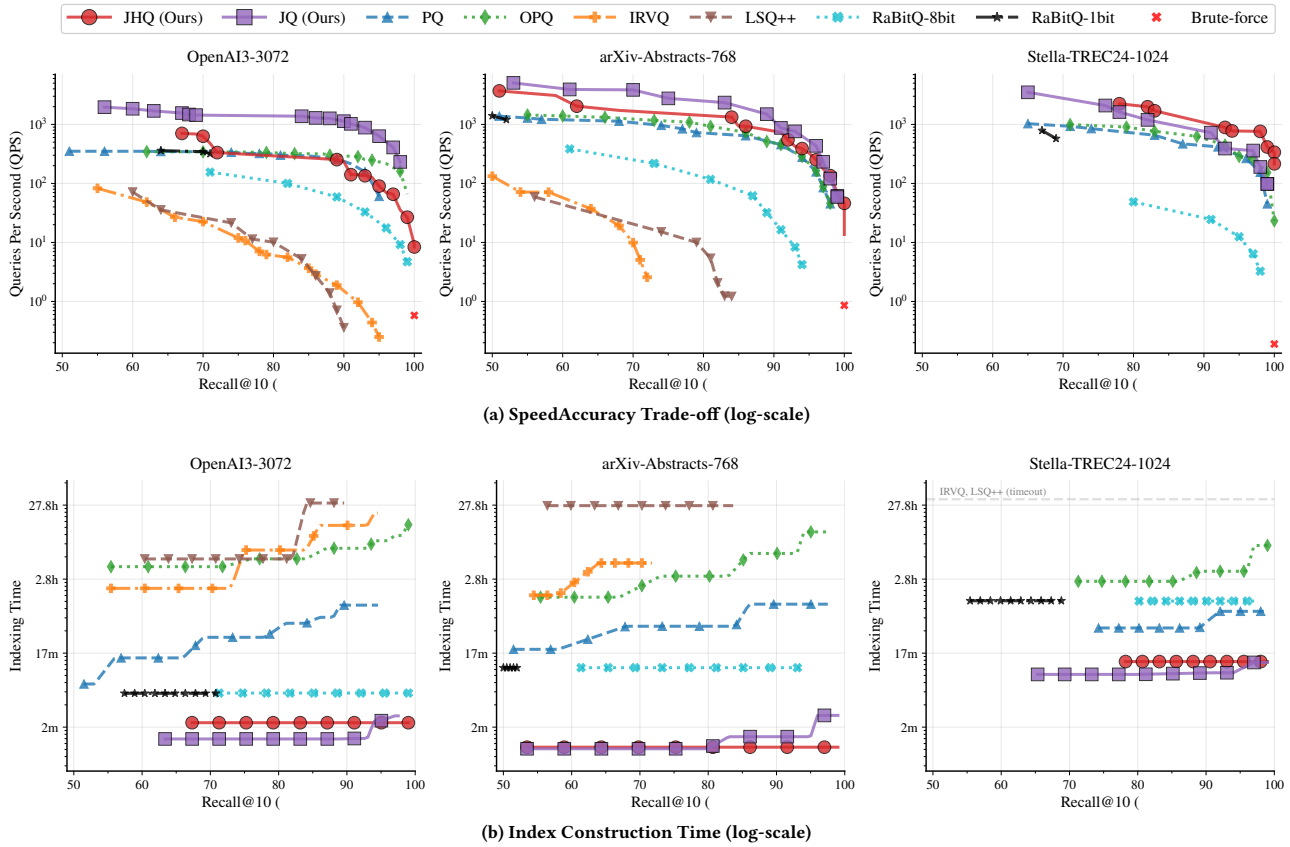


Figure 4: Efficiency evaluation across three datasets: (a) QPS vs. recall@10; (b) index construction time.

which demonstrates their strong ability to handle such challenges. For example, on arXiv-Abstracts-768 with high LID and small RC, our JQ achieves 2289 QPS at 85% recall, which is 3.6× faster than PQ and 1413× faster than LSQ++. This is largely attributed to the dimensional independence achieved by the JL transformation leveraged in our algorithms.

Distance Error. In addition to speed–accuracy, we also experimentally evaluate the distance error across three benchmark datasets with consistent parameter settings: $D_s = 8$, $B = 8$, $B_r = 4$. This yields the same theoretical bounds 6.75×10^{-1} for JQ and 4.22×10^{-2} for JHQ on all datasets. For each dataset, we randomly sample 5,000 pairs of query and database vectors, compute the true and approximated distances, and report the empirical maximum distance errors. As shown in Table 4, the empirical maximum distance errors (Emp) range from 3.10×10^{-1} to 5.99×10^{-1} for JQ and from 5.6×10^{-3} to 2.26×10^{-2} for JHQ, which are even smaller than the theoretical bound (Thr), thereby validating Theorem 4 and 7.

Scalability. We test the scalability by uniformly sampling 100K, 1M, 5M, 10M vectors from the Stella-TREC24-1024 dataset, whose size is 17M. As can be seen from Figure 5, when the dataset size increases, the QPS of JQ degrades more than JHQ, which indicates that our JHQ excels in handling large datasets. For instance, at 90% recall, the QPS of JQ decreases from 3963 to 723 by a factor of 5.4, that of JHQ decreases from 2230 to 889 by a factor of 2.5.

5.2 Index Compactness and Efficiency

We report index storage and search-time memory bandwidth on two representative datasets, OpenAI3-3072 (high-dimensional) and Stella-TREC24-1024 (large-scale), for methods with ≥ 95% recall.

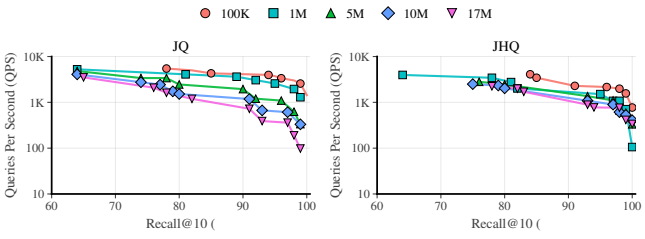


Figure 5: Scalability Evaluation

Index Storage. We report *Index Size* and *Index Compression Ratio (CR)* relative to the original vectors. As shown in Table 3, JQ achieves the highest index compression on both datasets (42× on OpenAI3-3072 and 30× on Stella-TREC24-1024). This is because JQ's bounded

Table 3: Comparison of index and code storage.

Dataset	Algorithm	Size (MB)	Index CR	ECL (bit)	Code CR
OpenAI3-3072	JQ	**278**	**42×**	1536	64×
	JHQ	3034	4×	**129**	**762×**
	PQ	425	28×	3072	32×
	OPQ	303	39×	2048	48×
	IRVQ	815	14×	6144	16×
	RaBitQ-8bit	3059	4×	24576	4×
Stella-TREC24-1024	JQ	**2327**	**30×**	1024	32×
	JHQ	9922	7×	**512**	**64×**
	PQ	4493	15×	2048	16×
	OPQ	**2323**	**30×**	1024	32×
	RaBitQ-8bit	17796	4×	8192	4×

quantization error allows us to use shorter codes while maintaining the same recall, thus reducing the required code length. In contrast, JHQ has a lower index CR (4×–7×) because it additionally stores residual-level codes, trading some index compactness for the much lower search-time memory traffic discussed next.

Query-Time Memory Utilization. We use the *Effective Code Length (ECL)* to quantify the average number of bits read from memory per candidate during querying. Smaller ECL means fewer bits fetched, better memory utilization, and higher query efficiency. For JHQ, ECL $= L_p + (\alpha k/n)\, L_r$, where L_p and L_r are the primary and residual code lengths, k is the number of retrieved neighbors, n is the candidate set size, and α is the refinement factor. For other methods, ECL is the length of the quantized code per candidate. JHQ attains the smallest ECL on both datasets, accessing only 129 bits on OpenAI3-3072 (12× lower than JQ) and 512 bits on Stella-TREC24-1024 (2× lower than JQ). This is because JHQ uses shorter primary codes for all candidates and reads residual codes only for promising ones, which reduces query-time memory access.

Index Build Efficiency. Figure 4b reports index construction time on the same three datasets. Appendix B.2 includes the remaining datasets and confirms the same qualitative conclusions. Our methods, JQ and JHQ, achieve the fastest index construction with substantial time reduction across all datasets. It is mainly because the JL transformation's dimensional independence enables quick Lloyd-Max centroid computation in $O(K)$ time per subspace, while other methods like PQ require expensive k-means clustering with $O(InK)$. Also, JHQ takes 1–5 more minutes than JQ since the residual level introduces additional overhead. For example, JQ provides a 62×–102× speedup in index construction compared to OPQ, because OPQ needs $O(d^3)$ time to train the rotation matrix. JHQ is 152× to 1800× faster than LSQ++ and JQ is 596× to 3600× faster than LSQ++, since training the LSQ++ codebook is NP-hard.

5.3 Ablation Study and Sensitivity Analysis

We ablate the JL transformation and the hierarchical design on two representative datasets: OpenAI3-3072 and Stella-TREC24-1024. Additional ablations on the remaining datasets are deferred to Appendix B.3 and are consistent with the conclusions drawn here. Finally, we present a focused case study on OpenAI3-3072 comparing scalar and vector quantization.

Impact of JL Transformation. As shown in Figure 6(a,b), JQ with JL outperforms the variant without JL on Stella-TREC24-1024 with higher QPS and recall, and it also dominates on OpenAI3-3072 once recall exceeds about 80%; the no-JL variant is only slightly

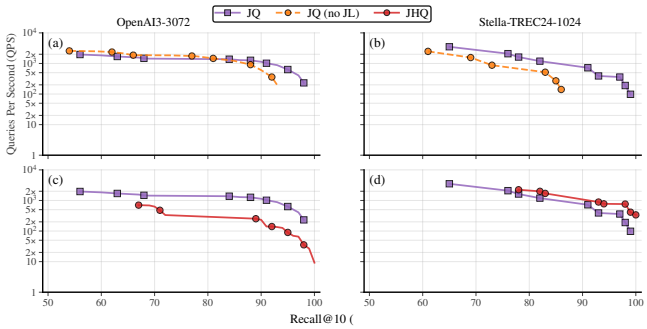


Figure 6: Ablation study. (a,b) JL transform; (c,d) hierarchical.

Table 4: Validation of distance error bounds.

Dataset	JQ		JHQ	
	Emp	Thr	Emp	Thr
OpenAI3-3072	0.318	0.675	0.0056	0.0422
arXiv-Abstracts-768	0.310	0.675	0.0107	0.0422
Stella-TREC24-1024	0.327	0.675	0.0093	0.0422

faster at very low recall because it skips the JL transformation. This confirms that the JL transformation, which yields a near-Gaussian representation, is key to efficient codeword computation.

Impact of Hierarchical Design. Figure 6(c,d) compares JQ with the hierarchical JHQ index. On the million-scale OpenAI3-3072 dataset, JQ is faster at fixed recall because residual refinement mainly adds overhead when primary codes are already accurate. On the 17M Stella-TREC24-1024 dataset, JHQ gives a better high-recall speed–accuracy trade-off by using short primary codes to prune candidates and refining only a small subset.

Scalar vs. Vector Quantization. On OpenAI3-3072, replacing JQs vector quantizer with independent per-dimension scalar quantizers preserves accuracy (recall > 99%) but reduces throughput: at 99% recall, scalar quantization is about 285× slower than JQ and nearly as slow as brute-force search. This justifies our use of vector quantization (Appendix B.3 shows the speed–accuracy curve).

Sensitivity Analysis of Refinement Factor α. α in JHQ controls the candidate set size for residual-level refinement. We evaluate $\alpha \in \{2.0, 4.0, 8.0\}$ and $\alpha_{\max} = N/k$ (no selective refinement) on three representative datasets, as shown in Figure 7. Appendix B.4 reports the remaining datasets and shows the same sensitivity pattern. Across all cases, $\alpha = 4.0$ yields the best speedaccuracy trade-off; for example, on OpenAI3-3072 at 90% recall, it achieves 253 QPS, compared to 153 and 160 for $\alpha = 2.0$ and $\alpha = 8.0$, respectively. Smaller α over-prunes candidates and may miss true neighbors, while larger α and $\alpha_{\max}$ introduce unnecessary residual distance

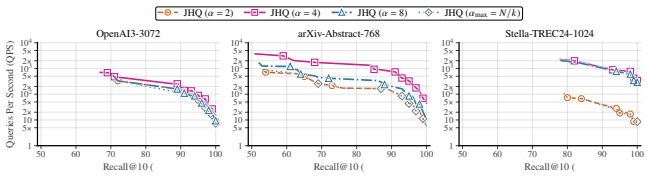


Figure 7: Sensitivity analysis on alpha.

computations with only marginal accuracy gains, confirming the effectiveness of our selective refinement design.

In summary, our evaluation validates that JQ and JHQ outperform state-of-the-art quantization baselines in speed-accuracy trade-offs, index compactness, and memory efficiency.

6 RELATED WORK

Approximate nearest neighbor (ANN) search algorithms can be broadly categorized into four paradigms: LSH-based methods [3, 4, 14, 19, 25, 30, 31, 39, 58–60], which map similar vectors to the same hash bucket with high probability for fast candidate retrieval through hashing. LSH offers simplicity and theoretical grounding but often at the cost of high memory usage and suboptimal recall in very high dimensions due to collision probabilities; VQ-based methods [7, 8, 13, 21, 22, 24, 33, 38, 42, 43, 47, 50, 67, 72] compress vectors into compact codes to reduce memory and enable efficient distance approximations. This makes VQ ideal for large-scale, high-dimensional datasets despite trade-offs in quantization accuracy, indexing time, and query speed that we discuss in detail below; tree-based methods [5, 9, 10, 16, 45, 53, 56] hierarchically partition the space (e.g., via kd-trees or random projection trees) to achieve logarithmic-time queries but degrade in high dimensions due to *the curse of dimensionality*, often necessitating randomization or ensembles; and graph-based methods [17, 18, 28, 32, 40, 41, 48] construct navigable graphs with nodes as vectors and edges connecting similar items to enable greedy traversal for high recall, though they typically incur high construction costs and memory overhead for dense graphs. Among them, VQ-based methods are particularly relevant to our work, as they excel in compressing dense vectors into a short code (i.e., a quantization code) while supporting fast distance computations. We analyze the VQ variants in depth next.

Product Quantization (PQ) [33] uniformly divides the original d-dimensional space into m subspaces, using k-means clustering within each subspace to learn n centroids for quantizing the corresponding sub-vectors. It supports fast ANN search with relatively small storage cost. To further reduce the storage, DeltaPQ [67] compresses the vectors using similarities between neighboring vectors without much accuracy loss. PQ assumes that distributions among different subspaces are mutually independent, but real high-dimensional data exhibit strong dependence between subspaces [34, 64]. Thus, its effectiveness degrades when this assumption fails [7, 24, 42]. To address this, optimized product quantization (OPQ) [24] decorrelates the subspaces with a learned rotation matrix before quantization – but it is computationally intensive to learn this rotation matrix ($O(d^3)$ to solve the Orthogonal Procrustes Problem [26], a substantial bottleneck in high dimensions). The alternatives (RVQ and AQ) avoid the subspace independence assumption by operating on full-dimensional vectors.

Residual Vector Quantization (RVQ) [13] introduces a hierarchical strategy that iteratively refines the encoding, where each stage quantizes the residual error from the previous one. However, RVQ's performance gains diminish quickly as more stages are added, particularly in high-dimensional spaces, because the residual vectors become increasingly random with reduced information entropy. To address this limitation, Improved RVQ (IRVQ) [38] enhances the encoding process by jointly optimizing all codebooks simultaneously using subspace clustering and multi-path encoding to maintain high information entropy across stages. Competitive Quantization (CompQ) [50] applies stochastic gradient descent (SGD) for iterative codebook updates through competitive learning. Residual Vector Product Quantization (RVPQ) [46] combines product and residual quantization by constructing residual structures within subspaces, but still requires expensive iterative training and complex beam search encoding. Our JHQ differs from RVQ in four key aspects: (i) JHQ adopts a two-level codebook, whereas RVQ uses a multi-stage one; (ii) the second-level codebook in JHQ is derived from scalar residuals, while RVQ employs full-dimensional residuals; (iii) JHQ obtains centroids via the Cartesian product of Lloyd-Max codewords, whereas RVQ uses k-means clustering; (iv) JHQ provides provable distance error bounds, which lead to a better trade-off between accuracy and efficiency in our experiments, whereas RVQ does not provide such guarantees.

Additive Quantization (AQ) [7] further relaxes constraints by representing vectors as unconstrained combinations of multiple full-dimensional codewords. However, this transforms the encoding task into an NP-hard combinatorial optimization problem, requiring expectation-maximization-like joint optimization to minimize quantization error. Other variants like Composite Quantization (CQ) [72] and Tree Quantization (TQ) [8] introduce structural constraints to improve efficiency; Local Search Quantization (LSQ) [42] introduces an iterated local search for efficient encoding; LSQ++ [43] speeds up the convergence of LSQ with direct codebook updates via stochastic gradient descent (SGD). These methods outperform PQ in terms of recall by reducing the quantization error.

Recently, RaBitQ [22] and its extension [21] introduce quantization methods with theoretical error bounds derived from spatial geometric properties of randomly rotated vectors. Although we provide experimental comparisons, we note that such a geometric approach is, in a sense, orthogonal to our work. Matryoshka Representation Learning (MRL) [36] makes ANN search more efficient on certain embedding models, significantly saving computation and reducing index size. It is complementary to our work: MRL reduces the number of dimensions, while we reduce the bits per dimension.

Despite their advances, AQ and RVQ face significant practical limitations: quadratic query complexity due to cross-term calculations between codewords and computationally intensive encoding procedures, making them impractical for high-dimensional data where fast indexing and query processing are critical.

7 CONCLUSION

We introduced two novel vector quantization algorithms, JQ and JHQ, to resolve critical performance bottlenecks in ANN search: prolonged indexing times, slow queries, and poor scalability. Using the orthogonal JL transform, we transform data into a near-Gaussian distribution with independent dimensions, enabling fast codebook generation without expensive training. This foundation underpins JQ and its hierarchical extension JHQ, validated on 6 datasets including one with > 17 million vectors. Our methods reduce indexing times from hours to minutes and achieve 310× query speedups over SOTA, with a theoretical distance error bound. In future work, we will focus on on-disk indexes for much larger datasets.

REFERENCES

[1] Cecilia Aguerrebere, Ishwar Bhati, Mark Hildebrand, Mariano Tepper, and Ted Willke. 2023. Similarity search in the blink of an eye with compressed indices. *arXiv preprint arXiv:2304.04759* (2023).

[2] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2015. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 29–38.

[3] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51, 1 (2008), 117–122.

[4] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. *Advances in neural information processing systems* 28 (2015).

[5] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *arXiv preprint arXiv:1804.06829* (2018).

[6] Tony Assi. 2023. Vogue933k Embeddings. Hugging Face Datasets. https://huggingface.co/datasets/tonyassi/vogue933k-embeddings Accessed 11/10/25.

[7] Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 931–938.

[8] Artem Babenko and Victor Lempitsky. 2015. Tree quantization for large-scale similarity search and classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4240–4248.

[9] Erik Bernhardsson. 2013. Annoy. https://github.com/spotify/annoy. Accessed: 2025-08-13.

[10] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When is nearest neighbor meaningful?. In *International conference on database theory*. Springer, 217–235.

[11] Christian Böhm, Stefan Berchtold, and Daniel A Keim. 2001. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys (CSUR)* 33, 3 (2001), 322–373.

[12] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*. 2318–2335.

[13] Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate nearest neighbor search by residual vector quantization. *Sensors* 10, 12 (2010), 11259–11273.

[14] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[15] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *arXiv preprint arXiv:2401.08281* (2024).

[16] Ada Wai-chee Fu, Polly Mei-shuen Chan, Yin-Ling Cheung, and Yiu Sang Moon. 2000. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *The VLDB Journal* 9, 2 (2000), 154–173.

[17] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2021), 4139–4150.

[18] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461474.

[19] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 541–552.

[20] Walter Gander. 1980. Algorithms for the QR decomposition. *Res. Rep* 80, 02 (1980), 1251–1268.

[21] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2025. 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 (2025), 1–26.

[22] Jianyang Gao and Cheng Long. 2024. 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 (2024), 1–27.

[23] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2, 1 (2023).

[24] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.

[25] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*. 518529.

[26] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2012. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE transactions on pattern analysis and machine intelligence* 35, 12 (2012), 2916–2929.

[27] Google Research. 2020. ViT base patch16 224. Model Card. https://huggingface.co/google/vit-base-patch16-224 Accessed 2025-11-10.

[28] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.

[29] Junfeng He, Sanjiv Kumar, and Shih-Fu Chang. 2012. On the difficulty of nearest neighbor search. *arXiv preprint arXiv:1206.6411* (2012).

[30] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[31] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[32] Masajiro Iwasaki and Daisuke Miyazaki. 2018. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *arXiv preprint arXiv:1810.07355* (2018).

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

[34] Takuya Kataiwa, Cho Hakaze, and Tetsushi Ohki. 2025. Measuring Intrinsic Dimension of Token Embeddings. *arXiv preprint arXiv:2503.02142* (2025).

[35] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 39–48.

[36] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. 2022. Matryoshka representation learning. *Advances in Neural Information Processing Systems* 35 (2022), 30233–30249.

[37] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[38] Shicong Liu, Hongtao Lu, and Junru Shao. 2015. Improved residual vector quantization for high-dimensional approximate nearest neighbor search. *arXiv preprint arXiv:1509.05195* (2015).

[39] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*. 950–961.

[40] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[41] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[42] Julieta Martinez, Joris Clement, Holger H Hoos, and James J Little. 2016. Revisiting additive quantization. In *European Conference on Computer Vision*. Springer, 137–153.

[43] Julieta Martinez, Shobhit Zakhmi, Holger H Hoos, and James J Little. 2018. Lsq++: Lower running time and higher recall in multi-codebook quantization. In *Proceedings of the European conference on computer vision (ECCV)*. 491–506.

[44] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. 2018. A survey of product quantization. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 2–10.

[45] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11, 1 (2002), 28–46.

[46] Lushuai Niu, Zhi Xu, Longyang Zhao, Daojing He, Jianqiu Ji, Xiaoli Yuan, and Mian Xue. 2023. Residual vector product quantization for approximate nearest neighbor search. *Expert Systems with Applications* 232 (2023), 120832.

[47] Mohammad Norouzi and David J Fleet. 2013. Cartesian k-means. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*. 3017–3024.

[48] Naoki Ono and Yusuke Matsui. 2023. Relative nn-descent: A fast index construction for graph-based approximate nearest neighbor search. In *Proceedings of the 31st ACM International Conference on Multimedia*. 1659–1667.

[49] OpenAI. 2024. text-embedding-3-large. Model Documentation. https://platform.openai.com/docs/models/text-embedding-3-large Accessed 2025-11-10.

[50] Ezgi Can Ozan, Serkan Kiranyaz, and Moncef Gabbouj. 2016. Competitive quantization for approximate nearest neighbor search. *IEEE Transactions on Knowledge and Data Engineering* 28, 11 (2016), 2884–2894.

[51] Qdrant. 2024. DBPedia Entities OpenAI3 Text Embedding 3 Large 1536 1M. Hugging Face Datasets. https://huggingface.co/datasets/Qdrant/dbpedia-entities-openai3-text-embedding-3-large-1536-1M Accessed 2025-11-10.

[52] Qdrant. 2024. DBPedia Entities OpenAI3 Text Embedding 3 Large 3072 1M. Hugging Face Datasets. https://huggingface.co/datasets/Qdrant/dbpedia-entities-

openai3-text-embedding-3-large-3072-1M Accessed 2025-11-10.
[53] Parikshit Ram and Kaushik Sinha. 2019. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th acm sigkdd international conference on knowledge discovery & data mining*. 1378–1388.
[54] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2021. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *arXiv preprint arXiv:2112.01488* (2021).
[55] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*. Springer, 291–324.
[56] Tomáš Skopal, Jaroslav Pokorný, and Václav Snášel. 2005. Nearest Neighbours Search using the PM-tree. In *International Conference on Database Systems for Advanced Applications*. Springer, 803–815.
[57] Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A Smith, Luke Zettlemoyer, and Tao Yu. 2022. One embedder, any task: Instruction-finetuned text embeddings. *arXiv preprint arXiv:2212.09741* (2022).
[58] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proceedings of the VLDB Endowment* (2014).
[59] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2009. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 563–576.
[60] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2010. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Transactions on Database Systems (TODS)* 35, 3 (2010), 1–46.
[61] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=wCu6T5xFjeJ
[62] The Alexandria Index. 2023. arXiv Abstracts Dataset. Hugging Face Datasets. https://huggingface.co/datasets/macrocosm/arxiv_abstracts Accessed 2025-11-10.
[63] The Information Engineering Lab. 2024. Stella TREC24 BioGen Embedding. Hugging Face Datasets. https://huggingface.co/datasets/ielabgroup/stella_trec24_biogen_embedding Accessed 2025-11-10.
[64] Hayato Tsukagoshi and Ryohei Sasano. 2025. Redundancy, Isotropy, and Intrinsic Dimensionality of Prompt-based Text Embeddings. *arXiv preprint arXiv:2506.01435* (2025).
[65] Upstash. 2024. Wikipedia 2024-06 BGE-M3. Hugging Face Datasets. https://huggingface.co/datasets/Upstash/wikipedia-2024-06-bge-m3 Accessed 2025-11-10.
[66] Cédric Villani. 2009. The wasserstein distances. In *Optimal transport: old and new*. Springer, 93–111.
[67] Runhui Wang and Dong Deng. 2020. DeltaPQ: lossless product quantization code compression for high dimensional similarity search. *Proceedings of the VLDB Endowment* 13, 13 (2020), 3603–3616.
[68] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, Vol. 98. 194–205.
[69] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. 2020. Visual Transformers: Token-based Image Representation and Processing for Computer Vision. arXiv:2006.03677 [cs.CV]
[70] Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. 2025. TurboQuant: Online Vector Quantization with Near-optimal Distortion Rate. *arXiv preprint arXiv:2504.19874* (2025).
[71] Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. 2024. Jasper and stella: distillation of sota embedding models. *arXiv preprint arXiv:2412.19048* (2024).
[72] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite quantization for approximate nearest neighbor search. In *International Conference on Machine Learning*. PMLR, 838–846.