The RaBitQ Library

Jianyang Gao [*1] **Yutong Gou** [*1] **Yuexuan Xu** [1] **Jifan Shi** [1] **Zhonghao Yang** [1] **Cheng Long** [1]

Abstract

High-dimensional vectors serve as a bridge between massive real-world data and AI applications such as large language models (LLMs). Yet, the data is often of large scale and with high dimensionality, which causes high costs in both memory consumption and computation. To reduce both costs, vector quantization has been widely adopted. A recent advance in vector quantization is the RaBitQ method, which supports flexible compression rates and provides asymptotically optimal error bounds for distance estimation. It demonstrates strong empirical performance in terms of space, accuracy and runtime. In this paper, we first provide a comprehensive overview of RaBitQ's design, insights and theoretical guarantees. We then introduce the RaBitQ Library, which covers some key implementation techniques of RaBitQ and its integration with indices for approximate nearest neighbor search.

1. Introduction

Artificial Intelligence (AI) has rapidly transformed many facets of modern life, from revolutionizing industries to influencing how we interact with technology on a daily basis. For example, large language models (LLMs) (Brown et al., 2020), which corresponds to a powerful type of AI application, has been used for many tasks including but not limited to question answering, code generation, and autonomous AI agents. These applications are fundamentally powered by diverse and massive data - including images, text, audio, video, and code and social media interactions.

To exploit the massive data, deep neural network models first extract their semantic information and represent it as high-dimensional vectors. The semantic similarity in the original data translates into geometric proximity in vector space. These vectors are then utilized in various downstream systems, e.g., they can be stored in vector databases (Wang et al., 2021) for tasks such as information retrieval (Santhanam et al., 2021), and retrieval-augmented generation (Gao et al., 2024b), or they can be directly fed to machine learning models to perform learning and generation tasks.

In the context of modern AI applications, high-dimensional vector data is often of large scale and with high dimensionality. For example, LLMs can take a prompt with millions of tokens as inputs, where each token would be embedded into a vector, and OpenAI's text-embedding-3-large model [1] can produce vectors with up to 3,072 dimensions. The large scale and high dimensionality cause challenges in both storage and computation. For example, the space cost of storing 1 billion vectors each with 1,000 dimensions would be 4TB and the computation cost of finding approximate nearest neighbors among massive vectors is high since it usually involves many distance computations and each of computation involves thousands of arithmetic operations on floating-point values and is costly.

Vector quantization has been widely adopted for addressing the challenges (Simhadri et al., 2024). The major idea is to first construct a codebook, which consists of a certain number of vectors. It then quantizes a vector to be the closest vector in the codebook and store its index as the quantization code. Since a code is usually shorter than a vector, the space is reduced. In addition, the distance computations based on quantized vectors are often more efficient than on the original vectors, and thus the computation costs are reduced. Among those existing vector quantization methods, the scalar quantization (SQ) and product quantization (PQ) (Jegou et al., 2010) are two most widely used ones. However, SQ struggles to produce reasonable accuracy with high compression rates. PQ offers better effectiveness for high compression rates but lacks theoretical performance guarantees and incurs the overhead of frequent random memory access for computing distances.

Recently, we introduced a new vector quantization method called RaBitQ (Gao & Long, 2024; Gao et al., 2024a). Specifically, RaBitQ constructs a randomized codebook,

[*]Equal contribution [1]College of Computing and Data Science, Nanyang Technological University, Singapore. Correspondence to: Cheng Long <c.long@ntu.edu.sg>.

Proceedings of the 1st Workshop on Vector Databases at International Conference on Machine Learning, 2025.

[1]https://openai.com/index/new-embedding-models-and-api-updates/

which is transformed from vectors of D-dimensional B-bit unsigned integers, where D is the dimensionality and B is the bit-width per dimension. For example, when $B = 1$, RaBitQ corresponds to a binary quantizer and provides a 32x compression rate (i.e., each floating-point value in the original vector is quantized to be 1 bit for each dimension). Based on the randomized codebook, RaBitQ's distance estimator is carefully designed so that it is unbiased and has a theoretical error bounded. It is proved that RaBitQ's error bound is asymptotically optimal across different compression rates. Empirically, RaBitQ achieves superior trade-offs among space, accuracy and efficiency compared to SQ and PQ. Since its first release in May 2024, RaBitQ has been widely adopted in systems from the industry including VectorChord (TensorChord, 2024), VSAG (Zhong et al., 2025), Milvus (Zilliz, 2024), Faiss (Faiss, 2025), CockroachDB (CockroachDB, 2025), and Volcengine from ByteDance (Volcengine, 2025). ElasticSearch adopts RaBitQ with some minor changes (e.g., removing the random rotation) and gives the variant a new name "BBQ" (Trent, 2025).

Beyond the algorithmic advancements of RaBitQ, numerous implementation techniques further enhance its practical performance. To facilitate deployment of RaBitQ in real-world systems, we develop the RaBitQ Library, which is designed with a strong emphasis on usability and efficiency. To promote ease of use, the library provides a set of well-defined interfaces and multiple data formats that support RaBitQ's advanced features. For example, RaBitQ allows each quantization code to be split into two components: a binary code (consisting of the most significant bit) and an ex-code (consisting of the remaining bits). During querying, the binary code is accessed first to compute a coarse distance estimate. If higher accuracy is required, the ex-code is accessed incrementally to refine the result. For practical efficiency, the library includes optimized implementations of key RaBitQ components such as the rotator and quantizer. These implementations trade off some theoretical guarantees in favor of significantly improved runtime performance, while still maintaining strong empirical accuracy.

The remaining part of the paper is organized as follows. Section 2 introduces the key features, insights and theoretical guarantees about RaBitQ. Readers can refer to the original RaBitQ papers (Gao & Long, 2024; Gao et al., 2024a) for more technical details. Section 3 briefly introduces the implementation techniques adopted in the RaBitQ Library. Section 4 describes how RaBitQ is combined with IVF, HNSW and QG for vector search. Section 5 concludes the paper and discusses future research directions. The library is publicly available at `https://github.com/VectorDB-NTU/RaBitQ-Library`.

2. The RaBitQ Algorithm

RaBitQ (Gao & Long, 2024; Gao et al., 2024a) is a vector quantization algorithm, which offers asymptotically optimal trade-off between space and accuracy of distance estimation [2]. It can work as a drop-in replacement of (uniform) scalar quantization and binary quantization.

The key advantages of RaBitQ include

- **High Accuracy with Tiny Space** - RaBitQ offers promising accuracy under tiny space budgets such as 1 to 4 bits per dimension.
- **Fast Distance Estimation** - For 1-bit quantization, RaBitQ enables efficient distance estimation using bitwise operations or FastScan (André et al., 2017). For multi-bit quantization, it performs distance estimation through arithmetic operations on unsigned integers.
- **Theoretical Error Bounds** - RaBitQ guarantees an unbiased estimator with asymptotically optimal theoretical error bound.

2.1. Insights

A key factor contributing to the optimality of RaBitQ is its effective exploitation of a distinctive property in high-dimensional spaces known as the *concentration of measure* (Ledoux, 2001; Vershynin, 2018). There have been vast theoretical studies on this phenomenon over the past decades (Alon & Klartag, 2017; Johnson & Lindenstrauss, 1984; Larsen & Nelson, 2017; Freksen, 2021). In this paper, instead of discussing the principle in rigorous mathematics, we aim to present a simple example to build an intuitive understanding on the counter-intuitive phenomenon.

Consider the following scenario: we are interested in determining the value of the projection of a unit vector $\mathbf{x}$ onto a specific vector—for simplicity, let's choose the first coordinate of the vector, represented as $\mathbf{x}[0]$. However, due to certain constraints, we do not have direct access to this value. Instead, our goal is to estimate its approximate range. Since the vector $\mathbf{x}$ is a unit vector, and with no further information, the only inference we can make is that the value of $\mathbf{x}[0]$ must fall within the interval $[-1, 1]$.

Next, let us investigate the case, where the vector $\mathbf{x}$ follows the uniform distribution on the unit sphere. To better understand the behavior of $\mathbf{x}[0]$ in this case, we generate 10^5 random vectors following this distribution. The empirical distribution of $\mathbf{x}[0]$ is plotted in Figure 1. On the left panel, we illustrate the case for a 3-dimensional vector. This scenario is intuitive: for a unit vector, $\mathbf{x}[0]$ can take any value

[2] It supports the estimation of Euclidean distances, inner products and cosine similarities. For the ease of narrative, we refer to all these similarity metrics as distances in the paper.

within the range $[-1, 1]$. On the right panel, however, the situation becomes far less intuitive when the vector has 1,000 dimensions. While the theoretically possible range of $\mathbf{x}[0]$ remains $[-1, 1]$, the figure reveals a striking phenomenon: the values of $\mathbf{x}[0]$ are highly concentrated around 0. This unexpected behavior highlights a fundamental distinction between low-dimensional and high-dimensional spaces: in high-dimensional spaces, randomness (e.g., the uniform distribution of $\mathbf{x}$ on the unit sphere) can lead to surprisingly certainty (i.e., the concentration of $\mathbf{x}[0]$ around 0).

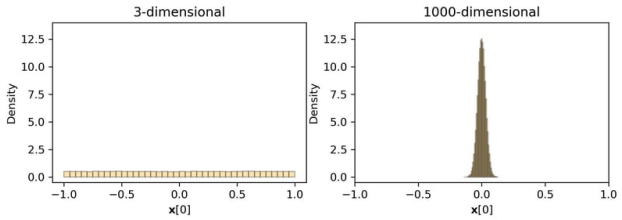

Figure 1. The Empirical Distribution of $\mathbf{x}[0]$ (Gao et al.).

This example typically demonstrates the phenomenon of concentration of measure in high-dimensional spaces. Formally, based on the seminal distributional Johnson-Lindenstrauss Lemma (Johnson & Lindenstrauss, 1984) (JL Lemma), with probability at least 99.9%, in this example, the absolute value of $\mathbf{x}[0]$ is bounded by $\Omega(1/\sqrt{D})$, where D is the dimensionality. For more theoretical studies around the phenomenon of concentration of measure, we refer readers to comprehensive surveys and textbooks (Ledoux, 2001; Vershynin, 2018; Freksen, 2021).

The concentration phenomenon in high-dimensional spaces leads to intriguing implications. In particular, in this example, when $\mathbf{x}$ has 3 dimensions, the only information we have about $\mathbf{x}[0]$ is that it lies within the interval $[-1, 1]$, which is not very informative. However, when $\mathbf{x}$ has 1,000 dimensions, the concentration phenomenon tells us that $\mathbf{x}[0]$ is highly unlikely to deviate from 0 by $\Omega(1/\sqrt{D})$. Since D is large, this interval becomes much narrower, providing a significantly tighter bound on $\mathbf{x}[0]$ in high-dimensional spaces. We note that this is particularly counter-intuitive because we did not access any single bit of data nor perform any computation to reach this conclusion. However, the uncertainty about $\mathbf{x}[0]$ significantly decreases. In other words, via analyzing the concentration phenomenon, we gain "free information" about $\mathbf{x}[0]$ without doing any computation.

This insight creates opportunities to enhance algorithms by leveraging this "free information". One area, where this can be especially advantageous is quantization. By effectively harnessing this phenomenon, we propose the RaBitQ algorithm (Gao & Long, 2024; Gao et al., 2024a) for vector quantization. In the algorithm, we construct a randomly rotated quantization codebook. The randomness in the codebook brings the aforementioned "free information", allowing us to construct an unbiased estimator for squared distances and inner products. In addition, in the RaBitQ papers (Gao et al., 2024a), we proved that RaBitQ's error bound is asymptotically optimal. To our knowledge, RaBitQ is the first practically deployable algorithm, which achieves the asymptotically optimal space-accuracy trade-off.

2.2. Theoretical Guarantees

Let $\mathbf{o}_r$ be a raw data vector, $\mathbf{q}_r$ be a raw query vector and $\mathbf{c}$ be a center vector, e.g., local centroids produced by KMeans clustering. Let $\mathbf{o} := \frac{\mathbf{o}_r - \mathbf{c}}{\|\mathbf{o}_r - \mathbf{c}\|}$ be the normalized data vector and $\mathbf{q} := \frac{\mathbf{q}_r - \mathbf{c}}{\|\mathbf{q}_r - \mathbf{c}\|}$ be the normalized query vector. RaBitQ first reduces the question of estimating squared distances or inner product between the raw vectors to the question of estimating the inner product of unit vectors as follows.

$$\begin{aligned}\|\mathbf{o}_r - \mathbf{q}_r\|^2 =& \|\mathbf{o}_r - \mathbf{c}\|^2 + \|\mathbf{q}_r - \mathbf{c}\|^2 \\ &- 2\|\mathbf{o}_r - \mathbf{c}\|\|\mathbf{q}_r - \mathbf{c}\| \cdot \langle \mathbf{o}, \mathbf{q} \rangle \qquad (1)\\ \langle \mathbf{o}_r, \mathbf{q}_r \rangle =& \langle \mathbf{o}_r, \mathbf{c} \rangle - \|\mathbf{c}\|^2 + \langle \mathbf{q}_r, \mathbf{c} \rangle \\ &+ \|\mathbf{o}_r - \mathbf{c}\|\|\mathbf{q}_r - \mathbf{c}\| \cdot \langle \mathbf{o}, \mathbf{q} \rangle \qquad (2)\end{aligned}$$

In these equations, the terms $\|\mathbf{o}_r - \mathbf{c}\|$ and $\langle \mathbf{o}_r, \mathbf{c} \rangle - \|\mathbf{c}\|^2$ only depend on the data vector and the center vector and can be pre-computed during indexing. The terms $\|\mathbf{q}_r - \mathbf{c}\|$ and $\langle \mathbf{q}_r, \mathbf{c} \rangle$ only depend on the query vector and the center vector. They can be computed once when a query comes and shared by many data vectors. With this, the question is reduced to the estimation of inner product between unit vectors. Note that the raw data vectors and query vectors can be any arbitrary vectors in the high-dimensional space. We do not assume that they are unit vectors. The theoretical results relevant to the unbiasedness and error bounds of RaBitQ are restated as follows.

Theorem 2.1 (Unbiasedness and Error Bounds)**.** *Let $\mathbf{o}$ be a unit data vector, $\mathbf{q}$ be a unit query vector, $\mathcal{C}_{rand}$ be a randomized codebook of unit vectors and $\hat{\mathbf{o}}$ be the quantized data vector of $\mathbf{o}$. An unbiased estimator is given by*

$$\mathbb{E}\left[\frac{\langle \hat{\mathbf{o}}, \mathbf{q} \rangle}{\langle \hat{\mathbf{o}}, \mathbf{o} \rangle}\right] = \langle \mathbf{o}, \mathbf{q} \rangle \qquad (3)$$

The following error bound of the estimator holds with probability at least $1 - 2\exp(-c_0 t^2)$.

$$\left|\frac{\langle \hat{\mathbf{o}}, \mathbf{q} \rangle}{\langle \hat{\mathbf{o}}, \mathbf{o} \rangle} - \langle \mathbf{o}, \mathbf{q} \rangle\right| \le \sqrt{\frac{1 - \langle \hat{\mathbf{o}}, \mathbf{o} \rangle^2}{\langle \hat{\mathbf{o}}, \mathbf{o} \rangle^2}} \cdot \frac{t}{\sqrt{D-1}} \qquad (4)$$

where t is a parameter which controls the failure probability. c_0 is a constant factor.

RaBitQ achieves asymptotic optimality on the space-accuracy trade-off established by a prior theoretical study (Alon & Klartag, 2017). The theorem about the optimality is restated as follows.

Theorem 2.2 (Optimality)**.** *Let B be the bit-width used per dimension. For $\epsilon > 0$ where $\frac{1}{\epsilon^2}\log\frac{1}{\delta} > D$, setting $B = \Theta\left(\log\left(\frac{1}{D}\cdot\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)\right)$ ensures that the error of inner product estimated by RaBitQ is bounded by ϵ with probability at least $1-\delta$.*

It is worth noting that the required bit-width B is *negatively* related to the dimensionality. In other words, for a fixed bit-width, higher dimensionality leads to lower error, which is counter-intuitive but aligns with the concentration of measure phenomenon.

3. Implementation Techniques

This section briefly introduces implementation techniques of RaBitQ in both indexing and querying phases. More detailed descriptions are left in the documentation due to the page limit. During indexing, the RaBitQ algorithm includes two steps: (1) Random Rotation - it samples a random rotation and applies it to all vectors; and (2) Quantization - it quantizes a vector of floating-point numbers into a vector of low-bit unsigned integers. During querying, the RaBitQ algorithm pre-processes the query vector with the same random rotation and estimates distances based on the rotated query vectors and the quantized data vectors. In this library, we provide efficient implementations for both components. Specifically, Section 3.1 and Section 3.2 present efficient implementation techniques of random rotation and quantization respectively. Recall that RaBitQ is used for estimating Euclidean distances and inner products. Section 3.3 illustrates data formats and estimator interfaces which ease the distance estimation. Section 3.4 and Section 3.5 briefly introduce the kernels for compact storage and computation respectively. Section 3.6 describes how to utilize RaBitQ's error bound for reranking.

3.1. Rotator

Random rotation (Johnson-Lindenstrauss Transformation, JLT in short) is a well-studied topic in the community of high-dimensional probability (Freksen, 2021). It originally refers to the random orthogonal transformation described in the seminal paper of Johnson-Lindenstrauss Lemma (Johnson & Lindenstrauss, 1984). Over the past decades, there has been a vast literature for fast JLT such as SRHT (Ailon & Chazelle, 2009) and Kac's Walk (Jain et al., 2020).

In the library, we adopt two algorithms for random rotation. One is the classical random orthogonal transformation (Johnson & Lindenstrauss, 1984), which takes $O(D^2)$ time for rotation. The other is an algorithm which combines SRHT (Ailon & Chazelle, 2009) and Kac's Walk (Jain et al., 2020), which takes $O(D\log D)$ time for rotation. The implementation of SRHT is taken from FFHT library (Andoni et al., 2015). In particular, the original SRHT supports only power-of-two dimensionalities. The combined approach enables efficient random rotation for more general dimensions. Due to the increasing dimensionality of vectors, we recommend use the latter, which runs faster and provides similar rotation quality.

3.2. Quantizer

After applying random rotation, the quantizer maps the rotated floating-point vectors to unsigned integer vectors. The library provides two implementations to accommodate different usage scenarios. The first implementation follows the algorithm detailed in the original RaBitQ paper (Gao et al., 2024a), achieving optimal accuracy at the cost of longer quantization time. The second implementation introduces approximations that slightly reduce accuracy but significantly accelerate quantization. Specifically, the original implementation of RaBitQ enumerates multiple rescaling factors. For each factor, it rescales the data vector and rounds it to the nearest vector in the codebook. Then it selects the rescaling factor and vector that minimize the quantization error. In the fast implementation, instead of enumerating factors, it directly uses the expected optimal rescaling factor [3] to perform the rounding and generates the quantization code.

Based on either of the implementations, the library provides a quantizer that can replace scalar quantization seamlessly. Specifically, for an input vector $\mathbf{x}$ and a specified bit-width B, the quantizer outputs a rescaling factor Δ_x, a shifting factor v_l [4] and a vector of B-bit unsigned integers $\mathbf{x}_u$ such that

$$\mathbf{x} \approx \Delta_x \mathbf{x}_u + v_l \mathbf{1}_D \tag{5}$$

where $\mathbf{1}_D$ denotes the all one vector in D-dimensional space.

3.3. Estimator Interface and Data Formats

Recall that RaBitQ is used for estimating Euclidean distances and inner products. In this part, we present the estimator interface and data formats adopted for supporting the estimation. Specifically, the library provides an interface, which receives the dimensionality D, the number of bits per dimension B, a floating-point vector $\mathbf{x}$, a center vector $\mathbf{c}$ and a flag which indicates the type of metrics (i.e., Euclidean distance or inner product) as inputs. It outputs a code vector $\mathbf{x}_u$ and several floating-point factors which are used for the distance estimation. It is worth highlighting that, with the

[3] The expected optimal rescaling factor is computed by averaging the optimal factors computed from several vectors, which are sampled from the uniform distribution on the unit sphere. Note that the factor is independent to the dataset.

[4] Note that storing v_l is optional, as it can be derived as $v_l = -\Delta_x \times \frac{2^B-1}{2}$.

new data formats and estimator interfaces, the query vector only needs to be pre-processed once during search. This significantly reduces the processing costs compared to the original implementation, which processes the query vector for each distinct center vector.

3.3.1. Basic Data Formats for Distance Estimation

The basic data format includes a code vector of B-bit unsigned integers $\mathbf{x}_u$ and three factors `F_add`, `F_rescale` and `F_error`. When a query comes, it prepares a randomly rotated raw query vector $\mathbf{q}'_r$ and three factors `G_add`, `G_kBxSumq` and `G_error` for every center vector. Let `ip` be the inner product between the code vector $\mathbf{x}_u$ and the rotated query vector $\mathbf{q}'_r$. Based on these factors, the estimated Euclidean distance or inner product can be easily computed as follows.

```
float est_dist = F_add + G_add + F_rescale
    * (ip + G_kBxSumq);
float error_bound = F_error * G_error;
float lb_dist = est_dist - error_bound;
float ub_dist = est_dist + error_bound;
```

It is worth highlighting that this implementation works for both similarity metrics - the detailed values of the factors would be different for different metrics.

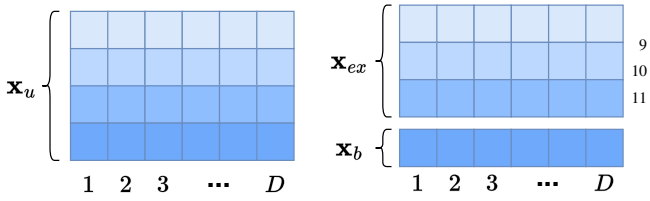

Figure 2. Splitting quantization codes into two parts (Gao et al., 2024a).

3.3.2. Advanced Data Formats for Incremental Distance Estimation.

RaBitQ supports an advanced feature known as incremental distance estimation. As illustrated in Figure 2, each code vector $\mathbf{x}_u$ can be decomposed into two components: the binary code $\mathbf{x}_b$ (comprising the most significant bits) and the ex-code $\mathbf{x}_{ex}$ (the remaining bits). During ANN querying, the algorithm can first estimate a coarse distance using only the binary code. If higher accuracy is needed, it can then access the ex-code to refine the estimate. This enables a flexible trade-off between efficiency and accuracy. It is important to distinguish this from a commonly used two-stage filtering strategy, where coarse distances (e.g., from 1-bit codes) are used to filter candidates, followed by re-ranking with a separate distance computation (e.g., using 4-bit codes). Although both approaches use a total of 5 bits, the two-stage strategy achieves the accuracy of 4-bit quantization, whereas incremental distance estimation achieves the accuracy of 5-bit quantization—effectively halving the error.

An advanced data format is provided to ease the incremental distance estimation. Specifically, the format stores the binary code $\mathbf{x}_b$ and the ex-code $\mathbf{x}_{ex}$ are separately. Three factors `F_add`, `F_rescale` and `F_error` are prepared for the binary codes and another three factors `F_add_ex`, `F_rescale_ex` and `F_error_ex` are provided by the ex-codes. When a query comes, it prepares a randomly rotated raw query vector $\mathbf{q}'_r$ and four factors `G_add`, `G_k1xSumq` and `G_kBxSumq` and `G_error` for every center vector. The detailed definition of the factors is left in the documentation of the library due to the limit of space. Let `ip_bin` be the inner product $\langle \mathbf{x}_b, \mathbf{q}'_r \rangle$ and `ip_ex` be the inner product $\langle \mathbf{x}_{ex}, \mathbf{q}'_r \rangle$. The incremental distance computation can be performed in the following way.

```
// 1-bit dist
float est_dist = F_add + G_add + F_rescale
    * (ip_bin + G_k1xSumq)
float bound = F_error * G_error
float ub_dist = est_dist + bound
float lb_dist = est_dist - bound

// boost to full-bit dist
float ex_est_dist = F_add_ex + G_add +
    F_rescale_ex * (ip_bin * (1 << (B-1)) +
     ip_ex + G_kBxSumq)
float ex_bound = F_error_ex * G_error
float ex_ub_dist = ex_est_dist + ex_bound;
float ex_lb_dist = ex_est_dist - ex_bound;
```

3.4. Kernels for Storing Codes Compactly

Recall that RaBitQ offers a key advantage: it achieves high accuracy even with small bit-widths, particularly when $B < 8$. However, encoding each coordinate with fewer than 8 bits leads to storage misalignment with standard byte boundaries. This misalignment poses challenges for both compact representation and efficient access, necessitating a specialized storage format that minimizes memory overhead while supporting high-speed operations. Furthermore, since the quantized codes are used to compute inner products with floating-point query vectors, the storage format must enable fast packing and unpacking to avoid becoming a bottleneck during distance estimation. Efficient decoding is particularly critical for exploiting SIMD-based acceleration.

In practice, modern applications often adopt vector dimensionalities that are multiples of 64—such as 512, 768, 1024, or 1536—due to design choices in deep learning models. To leverage this regularity, we design a compact storage scheme that efficiently supports 1-bit to 8-bit quantization over blocks of 64 coordinates. Figure 3 illustrates an example for 2-bit quantization. The codes for a 64-dimensional

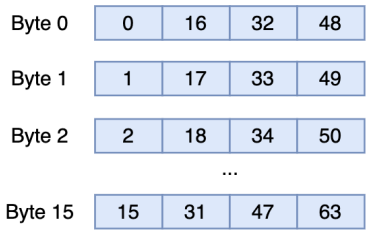

Figure 3. The Compact Storage of 2-bit Quantization.

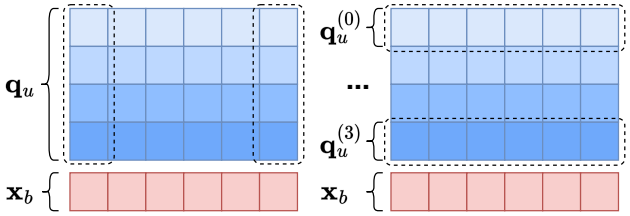

Figure 4. Bitwise Decomposition of Query Vectors (Gao & Long, 2024).

vector are stored in a byte array of length 16. Each row in the figure represents one byte. Specifically, the 0-th byte encodes the 2-bit codes for the 0-th, 16-th, 32-th, and 48-th dimensions. The 1st byte encodes the 2-bit codes for the 1st, 17th, 33rd, and 49th dimensions, and so on. This layout enables efficient unpacking using SIMD operations such as bit shifting and masking, which can be implemented using SSE instructions. The storage formats for other bit-widths follow similar principles and are documented in detail in the library's documentation.

3.5. Kernels for Computing Inner Products

Computing the inner product between a code vector (an array of unsigned integers) and a floating-point vector is a core operation in RaBitQ. The library provides vectorized kernels for this task. The kernels are categorized based on the bit-width of the codes:

- **1-bit Codes**: Two implementations are provided. The first leverages bitwise operations to efficiently estimate inner products for individual codes. The second employs FastScan (André et al., 2017), which processes inner products in batches.
- **Multi-bit Codes**: The codes are unpacked into vectors of 8-bit unsigned integers, enabling the inner product computation using standard vectorized operations.

Bitwise Operations. For computing inner products based on bitwise operations, we further quantize query vectors into vectors of unsigned integers. Let B_q be the bit-width used for quantizing query vectors. By default, we set $B_q = 4$ and use the fast implementation of RaBitQ in Section 3.2 to quantize query vectors. As is illustrated in Figure 4, the query vector is further decomposed into B_q binary vectors. Finally, through bitwise-and and popcount operations on the binary vectors, we can compute the inner product between the 1-bit codes and rotated query vectors approximately. Note that the bit-width used for quantizing query vectors is clearly larger than that for data vectors. Consequently, the overall error remains primarily determined by the quantization error of the data vectors.

FastScan. FastScan (André et al., 2017) is originally proposed to accelerate PQ's distance computation and has been widely adopted in real-world systems (Douze et al., 2024). It processes 32 codes in a batch, reorganizing their data layout during indexing. During querying, it quantizes the PQ lookup tables (LUTs), loads them into registers, and computes estimated distances for a batch of codes simultaneously using vectorized operations. In the library, we integrates the implementation of FastScan from Faiss (Douze et al., 2024). We provide two versions of FastScan. The first quantizes each coordinate of the query vector to 6 bits, enabling LUT values to be stored as 8-bit unsigned integers. The second quantizes each coordinate to 14 bits, storing LUT values as 16-bit unsigned integers, which can then be decomposed into two 8-bit LUTs for further computation. Each version provides different accuracy levels based on the query vectors. For details, please refer to the original papers of RaBitQ and the documentation of the code repository.

3.6. Reranking

Reranking is a technique widely adopted for improving recall of vector search. During searching, ANN algorithms usually (1) shortlist a set of candidates based on an index and their quantization codes (e.g., in memory); (2) retrieve the raw vectors (e.g., from disks) for those candidates which have the smallest estimated distances; and (3) computes the exact distances for the candidates to find the nearest neighbors. RaBitQ has a unique advantage in reranking due to its theoretical error bound. It can skip reranking a candidate if the lower bound of its estimated distance is larger than the upper bound of the distance of the nearest neighbors.

4. RaBitQ for ANN

In the library, we provide reference implementations which combine RaBitQ with existing popular ANN indices including IVF (Jegou et al., 2010), HNSW (Malkov & Yashunin, 2020) and QG (Yahoo, 2018) respectively. Each corresponds to a different trade-off between memory consumption and query efficiency, i.e., from IVF to HNSW to QG, the memory consumption increases while the query efficiency usually improves. In both IVF and HNSW, RaBitQ

is employed to reduce memory consumption by storing quantization codes instead of raw vectors (i.e., arrays of 32-bit floating-point numbers). The library supports bit-widths ranging from 1 to 9 bits. In practice, 4-bit, 5-bit, and 7-bit quantization typically achieve 90%, 95%, and 99% recall, respectively—without accessing raw vectors for reranking. In QG, for each vertex of a graph-based index, it quantizes each of its neighbors' vectors with RaBitQ and computes the approximate distances of all its neighbors using FastScan efficiently. All implementations are built on the core operations of RaBitQ described in Section 3.

4.1. IVF+RaBitQ

The Inverted File Index (IVF) (Jegou et al., 2010) is a lightweight index with minimal memory consumption. During indexing, it applies KMeans to partition a set of data vectors into several clusters. During querying, for a query, it retrieves data vectors from the clusters whose centroids are nearest to the query. Then it computes the distances of these vectors and finds out the NN. IVF is usually combined with quantization, e.g., product quantization, in ANN (Douze et al., 2024) - it estimates distances based on quantization codes instead of computing distances exactly. In RaBitQ Library, we use a similar strategy to combine IVF with RaBitQ.

Index Phase. During indexing, we first build IVF with the KMeans clustering in Faiss (Douze et al., 2024). We randomly rotate all data vectors and centroids based on the rotator described in Section 3.1. Then in each cluster, we take its centroid as the center vector and quantize every data vector into the data formats for incremental distance estimation (Section 3.3.2). Recall that the format includes a binary code and an ex-code for every vector. We pack every 32 binary codes in a batch and reorganize their data layout such that FastScan can be applied to accelerate querying.

Query Phase. To find the NN for a given query, we first rotate the query vector, prepare the look-up tables for FastScan and pre-process factors for distance estimation. Then, we compute the distances between the query and the clusters' centroids to select the nearest n_{probe} (set by users) clusters to search. After that, we scan the clusters one by one to estimate distances using codes of RaBitQ. For each cluster, we first access the 1-bit codes to compute a coarse approximate distance and its error bounds using FastScan. Then, we compare the lower bound of these estimated distances with the distance of the currently searched NN. If the lower bound is smaller than the distance of the NN, i.e., it may update the NN, we access the ex-codes to boost the accuracy and update the NN if its distance is smaller. Otherwise, we drop them directly. After scanning the selected n clusters, we obtain the NNs.

4.2. HNSW+RaBitQ

Graph-based indices are widely adopted in approximate nearest neighbor (ANN) search due to their superior efficiency. Among existing graph-based indices, the Hierarchical Navigable Small World (HNSW) graph is particularly popular in both academia and industry (Malkov & Yashunin, 2020).

Index Phase. We follow the standard HNSW insertion routine to build the graph incrementally. During indexing, we use the raw vectors to construct graphs. Once the graph is complete, the raw vectors are discarded. Each vector is then quantized using RaBitQ. Specifically, we first apply KMeans clustering to obtain n_c centroids (typically $n_c = 16$). Then for a data vector, we use its nearest centroid as the center vector and quantize it into the data formats for incremental distance estimation (Section 3.3.2).

Query Phase. During querying, we first rotate the query vector, prepare factors for distance estimation and quantize a query vector into 4-bit per dimension via the fast version of RaBitQ's scalar quantizer (Section 3.2). We then traverse the HNSW graph: in the upper layers, the estimated distances based on binary codes guide us to the entry point for the next layer. Upon reaching the base layer, we employ an adaptive strategy to retrieve NNs. During traversing the graph, for each neighbor, we first compute a 1-bit distance and its error bound using only the binary code. If the lower bound is smaller than the distance of the currently searched NN, we access the ex-code to compute a more accuracy distance and update the NN.

4.3. QG+RaBitQ (SymphonyQG)

Besides replacing the raw vectors with quantization codes, there is another framework, namely QG, which combines quantization and graph to minimize the random memory access in graph-based searching. The framework of QG was firstly proposed in the open-source library NGT by Yahoo Japan (Yahoo, 2018). Specifically, for every data vector, QG creates quantization codes for all its neighbors on a graph-based index and compactly stores them. The codes are further reorganized for FastScan (André et al., 2017). Thus, during querying, when visiting a vector's neighbors, the algorithm can largely eliminate random memory access and fully utilize FastScan to estimate distances. Based on the QG framework, we proposed a more symphonious combination of RaBitQ and graph-based indices, which is named SymphonyQG and achieves the state-of-the-art query performance (Gou et al., 2025). In this library, we further optimize the implementation of SymphonyQG based on the RaBitQ kernel mentioned in previous sections. Unlike IVF and HNSW, the combination of RaBitQ and QG is not designed to reduce memory usage, but rather to minimize

random memory accesses during graph-based querying and to accelerate the search process.

Index Phase. The indexing of QG follows an iterative pattern following DiskANN (Jayaram Subramanya et al., 2019). At the beginning, the graph-based index is randomly initialized. Then, in each iteration, for every vertex, the algorithm finds its candidate neighbors by querying the ANNs based on the graph-based index generated in the last iteration. The candidate neighbors are next pruned based on the RNG-base pruning rule (Fu et al., 2017). It is worth noting that the process of searching every data vector's ANN is accelerated with FastScan.

Query Phase. When a query comes, it is first processed to lookup tables (LUTs) for FastScan. Then we traverse the graph-based index in iterations. In each iteration of the search process, we identify the unvisited vector with the smallest estimated distance. We next compute its exact distance from the query vector and the estimated distances of the vertex's neighbors by FastScan. The NN that has been searched so far will be updated based on the exact distances.

5. Conclusion and Discussion

In this paper, we reviewed the RaBitQ method, including its insights and theoretical results. We then developed the RaBitQ Library, which incorporates optimized implementations and user-friendly interfaces to support advanced features of RaBitQ such as incremental distance estimation.

One future direction is to go towards more data-aware RaBitQ. For now, RaBitQ captures the data distribution to some extent in the normalization step (e.g., it uses the centroids of KMeans as center vectors for normalization), it is interesting to explore other alternatives with enhanced data-awareness.

Another future direction is to go towards wider applications of RaBitQ beyond vector search. One promising direction lies in neural network quantization. Recent work (Yang et al., 2025) suggests that RaBitQ, with its bit-level flexibility, can be tailored to quantize neural layers while preserving inference accuracy. Additionally, RaBitQ can be leveraged in key-value (KV) cache management for LLM inference. As transformer-based models like GPT and LLaMA rely heavily on KV caches to maintain context across long sequences, optimizing the storage and retrieval of these high-dimensional keys and values becomes crucial. RaBitQ's compression could enable more efficient KV cache storage, allowing for longer context windows or reduced memory usage in inference pipelines.

Acknowledgment

This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Award MOE-T2EP20221-0013 and Tier 1 Award (RG20/24)). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

References

Ailon, N. and Chazelle, B. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, 2009. doi: 10.1137/060673096. URL https://doi.org/10.1137/060673096.

Alon, N. and Klartag, B. Optimal compression of approximate inner products and dimension reduction. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 639–650, 2017. doi: 10.1109/FOCS.2017.65.

Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., and Schmidt, L. Practical and optimal lsh for angular distance. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 1225–1233, Cambridge, MA, USA, 2015. MIT Press.

André, F., Kermarrec, A.-M., and Le Scouarnec, N. Accelerated nearest neighbor search with quick adc. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, ICMR '17, pp. 159–166, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450347013. doi: 10.1145/3078971.3078992. URL https://doi.org/10.1145/3078971.3078992.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

CockroachDB. Cockroach. https://github.com/cockroachdb/cockroach, 2025.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. The faiss library, 2024. URL https://arxiv.org/abs/2401.08281.

Faiss. Faiss. https://github.com/facebookresearch/faiss, 2025.

Freksen, C. B. An introduction to johnson-lindenstrauss transforms. *CoRR*, abs/2103.00564, 2021. URL https://arxiv.org/abs/2103.00564.

Fu, C., Xiang, C., Wang, C., and Cai, D. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*, 2017.

Gao, J. and Long, C. Rabitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proc. ACM Manag. Data*, 2 (3), May 2024. doi: 10.1145/3654970. URL https://doi.org/10.1145/3654970.

Gao, J., Gou, Y., Xu, Y., Shi, J., Long, C., Wong, R. C.-W., and Palpanas, T. High-dimensional vector quantization: General framework, recent advances, and future directions. *Data Engineering*, pp. 3.

Gao, J., Gou, Y., Xu, Y., Yang, Y., Long, C., and Wong, R. C.-W. Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search, 2024a. URL https://arxiv.org/abs/2409.09913.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. Retrieval-augmented generation for large language models: A survey, 2024b. URL https://arxiv.org/abs/2312.10997.

Gou, Y., Gao, J., Xu, Y., and Long, C. Symphonyqg: Towards symphonious integration of quantization and graph for approximate nearest neighbor search. *Proc. ACM Manag. Data*, 3(1), February 2025. doi: 10.1145/3709730. URL https://doi.org/10.1145/3709730.

Jain, V., Pillai, N. S., Sah, A., Sawhney, M., and Smith, A. Fast and memory-optimal dimension reduction using kac's walk. *The Annals of Applied Probability*, 2020. URL https://api.semanticscholar.org/CorpusID:220514587.

Jayaram Subramanya, S., Devvrit, F., Simhadri, H. V., Krishnawamy, R., and Kadekodi, R. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf.

Jegou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Johnson, W. B. and Lindenstrauss, J. Extensions of lipschitz mappings into a hilbert space 26. *Contemporary mathematics*, 26:28, 1984.

Larsen, K. G. and Nelson, J. Optimality of the johnson-lindenstrauss lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 633–638. IEEE, 2017.

Ledoux, M. *The Concentration of Measure Phenomenon*. Mathematical surveys and monographs. American Mathematical Society, 2001. ISBN 9780821837924. URL https://books.google.com.sg/books?id=mCX_cWL6rqwC.

Malkov, Y. A. and Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020. doi: 10.1109/TPAMI.2018.2889473.

Santhanam, K., Khattab, O., Saad-Falcon, J., Potts, C., and Zaharia, M. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *arXiv preprint arXiv:2112.01488*, 2021.

Simhadri, H. V., Aumüller, M., Ingber, A., Douze, M., Williams, G., Manohar, M. D., Baranchuk, D., Liberty, E., Liu, F., Landrum, B., Karjikar, M., Dhulipala, L., Chen, M., Chen, Y., Ma, R., Zhang, K., Cai, Y., Shi, J., Chen, Y., Zheng, W., Wan, Z., Yin, J., and Huang, B. Results of the big ann: Neurips'23 competition, 2024. URL https://arxiv.org/abs/2409.17424.

TensorChord. Vectorchord. https://github.com/tensorchord/VectorChord, 2024.

Trent, B. Better binary quantization (bbq) in lucene and elasticsearch. https://www.elastic.co/search-labs/blog/better-binary-quantization-lucene-elasticsearch, 2025.

Vershynin, R. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2018. doi: 10.1017/9781108231596.

Volcengine. Using diskann vector engine. https://www.volcengine.com/docs/6465/1553583, 2025.

Wang, J., Yi, X., Guo, R., Jin, H., Xu, P., Li, S., Wang, X., Guo, X., Li, C., Xu, X., Yu, K., Yuan, Y., Zou, Y., Long, J., Cai, Y., Li, Z., Zhang, Z., Mo, Y., Gu, J., Jiang, R., Wei, Y., and Xie, C. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD

'21, pp. 2614–2627, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383431. doi: 10.1145/3448016.3457550. URL https://doi.org/10.1145/3448016.3457550.

Yahoo. Neighborhood graph and tree for indexing high-dimensional data. https://github.com/yahoojapan/NGT, 2018. Accessed: 17 Apr, 2024.

Yang, Y., Gao, J., and Hu, W. Raana: A fast, flexible, and data-efficient post-training quantization algorithm. *arXiv preprint arXiv:2504.03717*, 2025.

Zhong, X., Li, H., Jin, J., Yang, M., Chu, D., Wang, X., Shen, Z., Jia, W., Gu, G., Xie, Y., Lin, X., Shen, H. T., Song, J., and Cheng, P. Vsag: An optimized search framework for graph-based approximate nearest neighbor search, 2025. URL https://arxiv.org/abs/2503.17911.

Zilliz. Knowhere. https://github.com/zilliztech/knowhere, 2024.