Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search

JIANYANG GAO, Nanyang Technological University, Singapore
YUTONG GOU, Nanyang Technological University, Singapore
YUEXUAN XU, Nanyang Technological University, Singapore
YONGYI YANG, University of Michigan, USA
CHENG LONG*, Nanyang Technological University, Singapore
RAYMOND CHI-WING WONG, The Hong Kong University of Science and Technology, Hong Kong

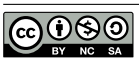
Approximate nearest neighbor (ANN) query in high-dimensional Euclidean space is a key operator in database systems. For this query, quantization is a popular family of methods developed for compressing vectors and reducing memory consumption. Among these methods, a recent algorithm called *RaBitQ* achieves the state-of-the-art performance and provides an asymptotically optimal theoretical error bound. RaBitQ uses 1 bit per dimension for quantization and compresses vectors with a large compression rate. In this paper, we extend RaBitQ to compress vectors with flexible compression rates - it achieves this by using B bits per dimension for quantization with $B = 1, 2, ...$ It inherits the theoretical guarantees of RaBitQ and achieves the asymptotic optimality in terms of the trade-off between space and error bounds as to be proven in this study. Additionally, we present efficient implementations of the extended RaBitQ, enabling its application to ANN queries to reduce both space and time consumption. Extensive experiments on real-world datasets confirm that our method consistently outperforms the state-of-the-art baselines in both accuracy and efficiency when using the same amount of memory.

CCS Concepts: • **Theory of computation → Data structures and algorithms for data management**; *Random projections and metric embeddings*; • **Information systems → Information retrieval query processing**.

Additional Key Words and Phrases: Approximate Nearest Neighbor Search, Johnson-Lindenstrauss Transformation, Quantization

ACM Reference Format:
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. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 202 (June 2025), 26 pages. https://doi.org/10.1145/3725413

*Corresponding author.
Authors' Contact Information: Jianyang Gao, Nanyang Technological University, Singapore, jianyang.gao@ntu.edu.sg; Yutong Gou, Nanyang Technological University, Singapore, yutong003@e.ntu.edu.sg; Yuexuan Xu, Nanyang Technological University, Singapore, yuexuan001@e.ntu.edu.sg; Yongyi Yang, University of Michigan, USA, yongyi@umich.edu; Cheng Long*, Nanyang Technological University, Singapore, c.long@ntu.edu.sg; Raymond Chi-Wing Wong, The Hong Kong University of Science and Technology, Hong Kong, raywong@cse.ust.hk.



ACM 2836-6573/2025/6-ART202
https://doi.org/10.1145/3725413

1 Introduction

Nearest neighbor (NN) queries on vectors in high-dimensional Euclidean space are a fundamental operator in database systems [50, 59–61, 65, 73, 81], with a wide range of applications such as retrieval-augmented generation [28] and information retrieval [41, 64]. However, due to the curse of dimensionality [35, 79], performing exact NN queries on large-scale databases becomes impractical because of their long response time. To balance time and accuracy, researchers often turn to a relaxed alternative: the *approximate nearest neighbor (ANN) query* [15, 29, 38, 45, 51].

In order to respond to user queries with low latency and high accuracy, the majority of existing studies focus on the in-memory setting of ANN which assumes that a device has sufficiently large RAM and mainly targets to optimize the search performance (the time-accuracy trade-off) [6, 23, 33, 44, 45]. However, in real-world systems, main memory is a precious resource. For example, in database systems deployed on commodity PCs [49], clouds [32, 66, 73] and mobile devices [9, 13], memory consumption non-trivially affects the costs of services and the experience of users. Therefore, handling ANN queries with the minimal space usage while ensuring strong search performance (i.e., achieving competitive efficiency at 90%, 95% and 99% recall) has become a crucial challenge. In particular, the space consumption of answering ANN queries with an index is comprised of two parts, one for the vectors and the other for the index. The part for vectors usually dominates the overall consumption because in most applications, they are embeddings produced by deep neural networks [41, 52]. Each usually has hundreds or thousands of dimensions and is correspondingly represented by hundreds or thousands of floating-point numbers. Thus, many existing studies target to reduce the memory consumption by compressing the vectors [1, 8, 27, 29, 38, 47].

To address this need, a well-known line of studies called *quantization* has been proposed [1, 8, 29, 38, 47, 74]. During indexing, these methods first construct a *quantization codebook*. For each vector in the database, they find the nearest vector in the codebook as its *quantized vector*, which is represented and stored as a short *quantization code*. During querying, they estimate the distances between data and query vectors based on the quantization codes. Product quantization (PQ) and scalar quantization (SQ) are the most popular threads of quantization [1, 29, 38]. However, as has been widely reported [29, 38], PQ and its variants can only produce poor recall (e.g., <80%) due to their severely lossy compression (i.e., they mostly adopt a compression rate of ≥32x). To recover the recall, these methods usually adopt a re-ranking strategy [17, 74], which accesses the *raw* vectors and computes exact distances to find the NN. Yet, this entails to store the raw vectors in main memory [1], undermining their target of saving space. When these methods are applied with a moderate compression rate (e.g., 8x or 4x) in order to produce reasonable recall without re-ranking, they often struggle to deliver competitive performance in terms of both accuracy and efficiency (e.g., they fail to outperform the classical scalar quantization, as shown in [1] and in Section 5.2). On the other hand, SQ and its latest variant LVQ [1] usually fail to deliver reasonable accuracy when the compression rate is relatively large (32x or 16x).

A recent study proposes a new scheme of quantization called *RaBitQ*, which outperforms PQ and its variants both empirically and theoretically [27]. Its superior performance leads to its deployment in real-world systems such as VectorChord [69], VSAG [31], Milvus [85] and Faiss [20]. ElasticSearch adopts RaBitQ with some minor changes (e.g., removing the random rotation) and gives the variant a new name "BBQ" [70]. Empirically, compared with PQ, when using the same length of the quantization codes, the method produces more accurate distance estimation with

[1]Another thread of studies access the raw vectors stored on disks for re-ranking [37, 75], which saves memory by making a compromise in efficiency. We note that they are orthogonal to vector compression and are out of the scope of the current study.

less time consumption. Theoretically, RaBitQ guarantees that its distance estimation is *unbiased* and has an *asymptotically optimal error bound*. Despite this, RaBitQ has the limitation that it only supports to compress a vector with a *large* compression rate (i.e., it compresses a D-dimensional vector into a D-dimensional binary vector). In this case, without re-ranking, the method can hardly produce reasonable recall either [27]. Considering the promising performance of RaBitQ at a large compression rate, a natural question is how to extend the method to achieve *moderate* compression rates so as to avoid the re-ranking step which entails storing the *raw* vectors. Ideally, this extension should (1) maintain accurate distance estimation with both *unbiasedness* and *asymptotic optimality* concerning the trade-off between space and error bounds, and (2) enable efficient distance estimation.

Nevertheless, to achieve both desiderata, the extension requires non-trivial designs. Let D be the dimensionality of a vector. The original RaBitQ uses D bits to quantize a vector, which corresponds to a large compression rate of 32x. To achieve moderate compression rates, we use $(B \cdot D)$ bits to quantize a vector, where B is a small integer, e.g., $B = 4$. Correspondingly, we need to construct a codebook with $2^{B \cdot D}$ vectors. According to the RaBitQ paper [27], (1) the unbiasedness of the estimator holds on condition that the codebook is comprised of randomly rotated unit vectors; and (2) the efficient computation can be achieved because the vectors in the codebook can be represented by binary vectors. When $B = 1$, there exists a natural construction of the codebook, which satisfies both requirements - it constructs a codebook by randomly rotating the vertices of a hypercube which is nested on the unit sphere (see Section 2.2 for more details). However, when $B > 1$ (which is the case that we target in this paper), there is no such natural construction. To fill this gap, we developed an extension of RaBitQ, which constructs the codebook by shifting, normalizing and randomly rotating the vectors whose coordinates are *B-bit unsigned integers*. The rationale is that (1) the normalization and random rotation operations ensure that the codebook is composed of randomly rotated unit vectors. This design allows our method to inherit the *unbiasedness* and the *error bound* of RaBitQ; and (2) the quantization code can be represented by a D-dimensional vector of B-bit unsigned integers (correspondingly the size of the codebook is $2^{B \cdot D}$) and the distance estimation can be supported by their arithmetic operations without exhaustively decompressing the codes.

Furthermore, for the new quantization codebook above, we note that the task of quantizing a data vector (i.e., finding its nearest vector in the codebook) is not as easy as the original RaBitQ. Therefore, we propose a new efficient algorithm for this task during indexing. We prove that the extended RaBitQ achieves the asymptotic optimality in terms of the trade-off between space and error bounds for the estimation of inner product between unit vectors. Based on the experimental studies on real-world datasets (Section 5.2), we verify that with the same number of bits, the extended RaBitQ produces consistently better accuracy than the state-of-the-art methods across diverse compression rates.

In addition, we apply the extended RaBitQ to ANN queries in combination with the *inverted file index* (*IVF index*) [38]. Beyond trivially using it for distance estimation, we find that the efficiency of the method can be further improved. In particular, in a quantization code of the extended RaBitQ (a vector of B-bit unsigned integers), the concatenation of the most significant bits of the coordinates is exactly equal to the quantization code of the original RaBitQ (a vector of 0/1 values). Motivated by this, we split our quantization codes and *separately* store their most significant bits and the remaining bits. During querying, we first estimate a distance by accessing only the most significant bits (i.e., it produces exactly the estimated distance of the original RaBitQ). If the estimated distance is sufficiently accurate to decide that a data vector cannot be the NN, then we drop it. Otherwise, we access the remaining bits and incrementally estimate a distance with higher accuracy based

on the complete code. Because the distance estimation based on the most significant bits can be realized with a rather efficient SIMD-based implementation called *FastScan* [4] and the accuracy is sufficient for pruning many data vectors, this operation helps significantly improve the efficiency.

We summarize our major contributions as follows.

(1) We propose a new quantization method by extending RaBitQ to support flexible compression rates. It constructs the codebook via shifting, normalizing and randomly rotating vectors of B-bit unsigned integers. Based on the design, it inherits RaBitQ's unbiased estimator for distance estimation. In addition, we prove that our method achieves the asymptotic optimality in terms of the trade-off between space and error bounds of estimating inner product between unit vectors.
(2) We apply the extended RaBitQ to ANN queries and introduce the efficient implementation of first estimating a distance based on the most significant bits. When the accuracy is insufficient, we access the remaining bits to incrementally estimate a distance with higher accuracy.
(3) We conduct extensive experiments on real-world datasets, which show that (1) the extended RaBitQ provides more accurate distance estimation and higher recall than all the baselines on all the tested datasets when using the same number of bits. At a compression rate of about 6.4x and 4.5x, it stably produces over 95% and 99% recall respectively without accessing raw vectors for re-ranking; (2) its empirical performance of is well aligned with the theoretical analysis.

The remainder of the paper is organized as follows. Section 2 introduces the ANN query and preliminary techniques. Section 3 presents the extended RaBitQ method. Section 4 illustrates the application of the extended RaBitQ to the in-memory ANN search. Section 5 provides extensive experimental studies on real-world datasets. Section 6 discusses related work. Section 7 presents the conclusion and discussion.

2 Preliminaries

2.1 ANN Query

Consider that there is a database which stores N data vectors in the D-dimensional Euclidean space. The *nearest neighbor (NN) query* targets to find the nearest data vector from the database for a given query vector $\mathbf{q}$. Due to the curse of dimensionality, the query is often relaxed to the *approximate nearest neighbor (ANN) query*, which targets smaller time/space consumption by making a slight compromise on the accuracy (e.g., it targets to reach 90%, 95% or 99% recall). In addition, the ANN query is often extended to finding the K nearest data vectors. For the ease of narrative, we assume that $K = 1$ in the algorithm description, while we note that our methods can be trivially applied to the query with $K > 1$. We focus on the in-memory ANN. However, unlike most of the existing studies on in-memory ANN which assume that the raw data vectors are stored in RAM [6, 44, 45], our method targets to reduce the memory consumption by compressing the raw data vectors and storing only the *compressed vectors* in main memory. That is, we target the setting where the raw vectors *cannot* be accessed during querying so as to save the main memory consumption, and this setting is the same as the one studied in LVQ [1]. Milvus [73] and SingleStore-V [65] also adopt IVF+SQ without hosting raw vectors in main memory for re-ranking. Note that re-ranking is a vital step only when a large compression rate is applied, because without re-ranking, it can only produce poor recall (<80%). However, when vectors are quantized with a moderate compression rate, an algorithm can achieve promising recall (>90%, 95%, 99%) without re-ranking. In in-memory ANN systems, re-ranking requires to host all raw vectors in RAM, which consume significant memory and undermine the primary goal of vector compression.

2.2 The Quantization Method RaBitQ

A recent paper proposes a new quantization method called *RaBitQ* [27], which quantizes a D-dimensional real vector into a D-bit string. It provides an unbiased estimator of squared distances and guarantees that the estimator has an asymptotically optimal error bound, which always holds regardless of the data distribution.

Specifically, given a raw data vector $\mathbf{o}_r$ and a raw query vector $\mathbf{q}_r$, it first normalizes the vectors based on a vector $\mathbf{c}$ (e.g., the centroid of a set of data vectors). Let $\mathbf{o} := \frac{\mathbf{o}_r - \mathbf{c}}{\|\mathbf{o}_r - \mathbf{c}\|}$ and $\mathbf{q} := \frac{\mathbf{q}_r - \mathbf{c}}{\|\mathbf{q}_r - \mathbf{c}\|}$ be the *normalized* data and query vectors. The (squared) Euclidean distance between $\mathbf{o}_r$ and $\mathbf{q}_r$ can be expressed as follows.

$$\|\mathbf{o}_r - \mathbf{q}_r\|^2 = \|(\mathbf{o}_r - \mathbf{c}) - (\mathbf{q}_r - \mathbf{c})\|^2 \tag{1}$$

$$= \|\mathbf{o}_r - \mathbf{c}\|^2 + \|\mathbf{q}_r - \mathbf{c}\|^2 - 2 \cdot \|\mathbf{o}_r - \mathbf{c}\| \cdot \|\mathbf{q}_r - \mathbf{c}\| \cdot \langle \mathbf{q}, \mathbf{o} \rangle \tag{2}$$

Note that the distance $\|\mathbf{o_r} - \mathbf{c}\|$ can be pre-computed in the index phase and $\|\mathbf{q}_r - \mathbf{c}\|$ can be computed when a query comes and can be shared by many data vectors. Therefore, the computation of the distances between the raw vectors can be reduced to that of the inner product of their normalized vectors. Then, it focuses on estimating the inner product of the normalized vectors [2].

During the **index** phase, RaBitQ constructs a set C of all possible bi-valued unit vectors, each consisting of values of $+\frac{1}{\sqrt{D}}$ and $-\frac{1}{\sqrt{D}}$. Then, it randomly rotates all vectors in C by multiplying them with a random *orthogonal matrix* [40] (a type of Johnson-Lindenstrauss Transformation) to form a quantization codebook C_r. The process can be described with equations as follows.

$$C_r := \left\{ P\mathbf{x} \;\middle|\; \mathbf{x} \in C \right\}, \textit{where } C := \left\{ +\frac{1}{\sqrt{D}}, -\frac{1}{\sqrt{D}} \right\}^D \tag{3}$$

where P is a random orthogonal matrix [40]. Note that the codebook is solely determined by the random orthogonal matrix P since the set of bi-valued unit vectors is *pre-defined* and does not rely on the data. Thus, it maintains the codebook C_r conceptually only by sampling and storing the matrix P. For each data vector $\mathbf{o}$, it finds the nearest vector $\bar{\mathbf{o}}_0$ in C_r as its *quantized vector*. The quantized vector is represented and stored as a *quantization code* $\bar{\mathbf{x}}_b \in \{0,1\}^D$ (a D-bit string) - recall that each quantized vector has a corresponding bi-valued unit vector, denoted by $\bar{\mathbf{x}}_0$, in C. Specifically, we have $\bar{\mathbf{o}}_0 = P\bar{\mathbf{x}}_0 = P\left(\frac{2}{\sqrt{D}}\bar{\mathbf{x}}_b - \frac{1}{\sqrt{D}}\mathbf{1}_D\right)$ where $\mathbf{1}_D$ is the D-dimensional vector whose coordinates are all ones.

During the **query** phase, it constructs an unbiased estimator for the inner product. The estimator has a theoretical error bound. We restate the estimator and its bound as follows.

Lemma 2.1 (Restating Theorem 3.2 in [27]). *$\frac{\langle \bar{\mathbf{o}}_0, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}_0, \mathbf{o} \rangle}$ is an unbiased estimator of $\langle \mathbf{o}, \mathbf{q} \rangle$. With probability at least $1 - \exp(-c_0\epsilon_0^2)$, its error bound is presented as*

$$\left| \frac{\langle \bar{\mathbf{o}}_0, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}_0, \mathbf{o} \rangle} - \langle \mathbf{o}, \mathbf{q} \rangle \right| \le \sqrt{\frac{1 - \langle \bar{\mathbf{o}}_0, \mathbf{o} \rangle^2}{\langle \bar{\mathbf{o}}_0, \mathbf{o} \rangle^2}} \cdot \frac{\epsilon_0}{\sqrt{D-1}} \tag{4}$$

where c_0 is a constant and ϵ_0 is a parameter which controls the failure probability of the bound.

It is proven that using RaBitQ to quantize a D-dimensional vector to a D-bit string, the inner product $\langle \bar{\mathbf{o}}_0, \mathbf{o} \rangle$ is highly concentrated around 0.8 [27]. Thus, the above lemma indicates that for estimating the inner product of two D-dimensional unit vectors, it guarantees a probabilistic error bound of $O(1/\sqrt{D})$ with high probability, which achieves the asymptotic optimality [2]. In terms

[2]Without further specification, in this paper, by data and query vectors, we refer to their normalized vectors.

of the computation of the estimator, we note that $\langle \mathbf{o}, \bar{\mathbf{o}}_0 \rangle$ is independent of the query and can be pre-computed before querying. The computation of $\langle \mathbf{q}, \bar{\mathbf{o}}_0 \rangle$ can be conducted as follows.

$$\langle \mathbf{q}, \bar{\mathbf{o}}_0 \rangle = \left\langle \mathbf{q}, P\left(\frac{2}{\sqrt{D}}\bar{\mathbf{x}}_b - \frac{1}{\sqrt{D}}\mathbf{1}_D\right)\right\rangle \tag{5}$$

$$= \frac{2}{\sqrt{D}}\langle \mathbf{q}', \bar{\mathbf{x}}_b \rangle - \frac{1}{\sqrt{D}}\sum_{i=1}^{D}\mathbf{q}'[i] \tag{6}$$

Here, $\mathbf{q}'$ denotes $P^{-1}\mathbf{q}$ and $\mathbf{q}'[i]$ denotes the i-th dimension of the vector $\mathbf{q}'$; (5) plugs in the definition of $\bar{\mathbf{o}}$ and (6) applies P^{-1} on both sides of the inner product. Note that $\sum_{i=1}^{D}\mathbf{q}'[i]$ depends only on the query vector. Thus, its computation can be conducted once and shared by many data vectors. For the computation of $\langle \mathbf{q}', \bar{\mathbf{x}}_b \rangle$, [27] introduces two versions of implementation. One is based on a SIMD-based implementation called *FastScan* [4], which can efficiently compute the estimated distances for data vectors batch by batch. The other is based on bitwise operations, which supports to efficiently estimate distances for individual vectors. We refer readers to the original papers of RaBitQ [27] and FastScan [3–5] for more technical details and theoretical analysis. With all the proposed techniques, RaBitQ supports to unbiasedly estimate the inner product (and further unbiasedly estimate the squared distances) with both promising accuracy and efficiency.

3 The Extended RaBitQ Method

3.1 Motivations and Overview

For ANN query, to avoid storing raw vectors in RAM and reach promising recall at the same time, it is necessary to compress the vectors with a moderate compression rate. Although RaBitQ achieves an asymptotically optimal error bound when quantizing D-dimensional vectors into D-bit codes (which corresponds to a rather high compression rate), it is still unclear how it can use more bits in pursuit of higher accuracy. In [27], it presents a simple way to use more bits by padding the vectors. Specifically, it pads D-dimensional vectors to $(B \cdot D)$ dimensions with zeros, and thus, it can use $(B \cdot D)$ bits by directly applying RaBitQ. Based on Lemma 2.1 (Theorem 3.2 of the paper [27]), in this case, it guarantees an error bound of $O(1/(\sqrt{B} \cdot \sqrt{D}))$ with high probability. However, this result shows that the error decays very slowly with respect to the number of bits used, indicating that this simple extension is not very effective for a moderate compression rate.

Formally, we note that, with this simple extension, to guarantee an error bound ϵ with the failure probability of at most δ, it requires $\Theta\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ bits [3]. However, as has been proven by a theoretical study [2], to guarantee an error bound ϵ, it is sufficient (and also necessary) to use $\Theta\left(D\log\left(\frac{1}{D}\cdot\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)\right)$ bits when the target accuracy is high [4], i.e., when $\frac{1}{\epsilon^2}\log\frac{1}{\delta} > D$. Note that the required number of bits in this result is logarithmic to ϵ^{-2} while the one in the former result is linear to ϵ^{-2}. For instance, when $\frac{1}{\epsilon^2}\log\frac{1}{\delta} = B \cdot D$, the gap would be that between $\Theta(D\log B)$ and $\Theta(B \cdot D)$. This implies that when targeting higher accuracy with more bits, there is substantial room to improve from the simple extension of RaBitQ.

Motivated by the discussions above, in this paper, we extend RaBitQ to support moderate compression rates and achieve asymptotically optimal estimation and efficient computation at the same time. Next, we present the details of our extended RaBitQ method, including (1) how it quantizes data vectors (Section 3.2), (2) how it computes distance estimators (Section 3.3), and (3) its summary and theoretical results (Section 3.4).

[3]This can be derived from the result in Lemma 2.1 (Theorem 3.2 of the paper [27]) by plugging in the parameters of ϵ and δ.
[4]The result is restated directly from Theorem 4.1 in the theoretical study [2].

3.2 Quantizing Data Vectors

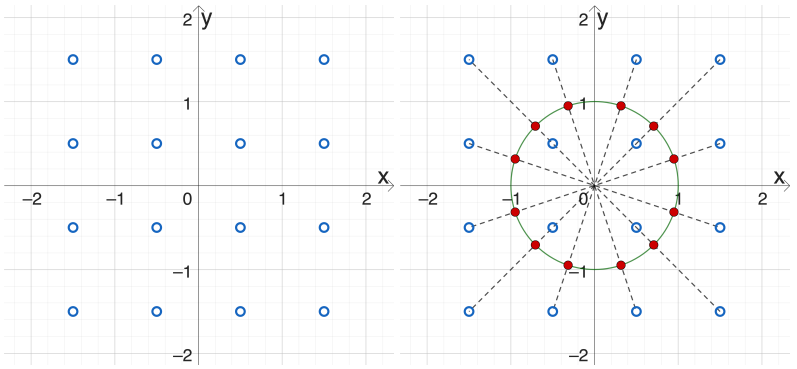


Fig. 1. This figure illustrates the quantization codebook of our method when $B = 2$ in the 2-dimensional space. The empty blue points in the left panel represent the set $\mathcal{G}$, i.e., a set of vectors on a uniform grid. The solid red points in the right panel represent the normalized vectors in $\mathcal{G}$. Applying a random rotation on the red points yields the codebook $\mathcal{G}_r$.

3.2.1 Constructing a Codebook. We notice that the simple extension in [27] provides sub-optimal accuracy because it ignores that the vectors can be hosted in a D-dimensional space. In contrast, it pads these vectors to $(B \cdot D)$ dimensions and applies RaBitQ in the $(B \cdot D)$-dimensional space. The codebook has the size of $2^{B \cdot D}$ and thus, it produces the quantization codes of $(B \cdot D)$ bits. Intuitively, given the same number of bits, a quantization algorithm would produce larger error when the vectors have higher dimensionality itself. Thus, it is likely that the sub-optimal accuracy is caused by the padding operation.

Instead of padding the vectors and constructing a codebook with $2^{B \cdot D}$ vectors in the $(B \cdot D)$-dimensional space, we consider constructing a codebook with $2^{B \cdot D}$ vectors in the D-dimensional space. Nevertheless, to inherit the unbiasedness and error bound of the estimator in RaBitQ and support efficient computation at the same time, the construction of the codebook is highly restrictive. **First**, the unbiasedness and the error bound (Lemma 2.1) hold on condition that the codebook is constructed by randomly rotating a set of *unit* vectors. **Second**, for efficient computation, it should better support to directly compute the inner product based on the quantization codes without an exhaustive decompression step. We propose the following codebook $\mathcal{G}_r$ to meet both requirements.

$$\mathcal{G} := \left\{ -\frac{2^B - 1}{2} + u \;\middle|\; u = 0, 1, 2, 3, ..., 2^B - 1 \right\}^D \tag{7}$$

$$\mathcal{G}_r := \left\{ P\frac{\mathbf{y}}{\|\mathbf{y}\|} \;\middle|\; \mathbf{y} \in \mathcal{G} \right\} \tag{8}$$

Specifically, as is illustrated in Figure 1, we first consider the set of vectors on the uniform grid $\mathcal{G}$. Then, we adopt their normalized vectors (i.e., $\frac{\mathbf{y}}{\|\mathbf{y}\|}$ in Equation (8)) and randomly rotate them by multiplying them with a random orthogonal matrix P) to form the codebook $\mathcal{G}_r$. The rationale is that (1) $\mathcal{G}_r$ is composed of randomly rotated unit vectors. Thus, it inherits the estimator in Lemma 2.1 along with its unbiasedness and error bound [5]; and (2) the vectors are generated by

[5]For the new codebook, Lemma 2.1 exactly holds. However, it is worth noting that the inner product between the data vector and the quantized data vector in the new codebook increases with respect to B. Thus, the error of the estimator decreases.

shifting, normalizing and rotating the vectors that consist of B-bit unsigned integers (Equation (7) and (8)). This enables efficient computation of inner product (see Section 3.3).

In practice, the construction of the codebook is very simple. We only need to sample a random transformation matrix P as RaBitQ does [27], and then the codebook $\mathcal{G}_r$ is determined. To store the codebook $\mathcal{G}_r$, we only need to store the sampled P, i.e., we maintain the codebook $\mathcal{G}_r$, which contains $2^{B\cdot D}$ vectors, conceptually. In particular, it is worth noting that when setting $B = 1$, the codebook is exactly the same as that of the original RaBitQ.

3.2.2 Computing the Quantization Codes of Data Vectors. Next, we present the algorithm of quantizing data vectors (i.e., finding their nearest vector in the codebook $\mathcal{G}_r$). Formally, for a vector $\mathbf{o}$, we target to find $\bar{\mathbf{o}} \in \mathcal{G}_r$ such that $\|\bar{\mathbf{o}} - \mathbf{o}\|^2$ is minimized. Let $\bar{\mathbf{y}}$ be the corresponding vector of $\bar{\mathbf{o}}$ in $\mathcal{G}$, i.e., $\bar{\mathbf{o}} = P\bar{\mathbf{y}}/\|\bar{\mathbf{y}}\|$. Following [27], we simplify the problem as follows.

$$\bar{\mathbf{y}} = \arg\min_{\mathbf{y}\in\mathcal{G}} \left\| P\frac{\mathbf{y}}{\|\mathbf{y}\|} - \mathbf{o} \right\|^2 = \arg\min_{\mathbf{y}\in\mathcal{G}} \left(2 - 2\left\langle P\frac{\mathbf{y}}{\|\mathbf{y}\|}, \mathbf{o} \right\rangle\right) \tag{9}$$

$$= \arg\max_{\mathbf{y}\in\mathcal{G}} \left\langle P\frac{\mathbf{y}}{\|\mathbf{y}\|}, \mathbf{o} \right\rangle = \arg\max_{\mathbf{y}\in\mathcal{G}} \left\langle \frac{\mathbf{y}}{\|\mathbf{y}\|}, P^{-1}\mathbf{o} \right\rangle \tag{10}$$

where Equation (9) is derived from the definition of $\bar{\mathbf{y}}$; Equation (10) applies an orthonormal matrix P^{-1} to both sides of the inner product. For conciseness, let us denote $\mathbf{o}' := P^{-1}\mathbf{o}$. Now the question reduces to one of finding $\bar{\mathbf{y}} \in \mathcal{G}$ such that the cosine similarity between $\bar{\mathbf{y}}$ and $\mathbf{o}'$ (Equation (10)) is maximized.

A natural idea is to enumerate the vectors in $\mathcal{G}$, compute the cosine similarity for each vector and find the vector $\bar{\mathbf{y}}$. However, enumerating all the vectors in $\mathcal{G}$ is not feasible as $\mathcal{G}$ contains $2^{B\cdot D}$ vectors. We observe that the enumeration can be significantly pruned based on the following lemma. The proof is left in the technical report [25] due to page limit.

Lemma 3.1. *Let* $\bar{\mathbf{y}} = \arg\max_{\mathbf{y}\in\mathcal{G}} \left\langle \frac{\mathbf{y}}{\|\mathbf{y}\|}, \mathbf{o}' \right\rangle$. *Then there exists a rescaling factor* $t > 0$ *such that* $\forall \mathbf{y} \in \mathcal{G}, \|t\cdot\mathbf{o}' - \bar{\mathbf{y}}\| \le \|t\cdot\mathbf{o}' - \mathbf{y}\|$.

Intuitively, this lemma indicates that for a vector $\mathbf{o}'$, if a vector $\bar{\mathbf{y}}$ has the largest *cosine similarity* from $\mathbf{o}'$ among the set $\mathcal{G}$, then there must be a re-scaling factor t such that $\bar{\mathbf{y}}$ also has the smallest *Euclidean distance* from the re-scaled vector $t\cdot\mathbf{o}'$ among the set $\mathcal{G}$. Note that $\mathcal{G}$ is a set of vectors on uniform grids, the nearest vector of $t\cdot\mathbf{o}'$ in $\mathcal{G}$ can be easily computed by rounding. This is to say, if we "enumerate every re-scaling factor t" and collect all the vectors obtained by rounding $t\cdot\mathbf{o}'$, then the optimal vector $\bar{\mathbf{y}}$ must be among the collected vectors. In practice, we do not need to enumerate every real value $t > 0$ because when two re-scaling factors are extremely close to each other, they would produce exactly the same vector after rounding. Thus, we only need to enumerate the critical values which change the results of rounding. Specifically, for a given t, assume that $t\cdot\mathbf{o}'[i]$ is currently rounded to a number x. Then, the next critical value of t that rounds $t\cdot\mathbf{o}'[i]$ to $(x+1)$ is $(x+0.5)/\mathbf{o}'[i]$. Since there are D dimensions and each dimension has 2^{B-1} different values after rounding [6], we will in total enumerate up to $(D\cdot 2^{B-1})$ critical values and correspondingly, up to $(D\cdot 2^{B-1})$ vectors in $\mathcal{G}$.

To efficiently implement this idea, we enumerate the critical values in an ascending order. The algorithm is presented in Algorithm 1. In this process, we dynamically maintain a vector $\mathbf{y}_{cur}$ - the vector obtained by rounding $t\cdot\mathbf{o}'$ based on the current t. We use two variables to maintain $\langle \mathbf{y}_{cur}, \mathbf{o}' \rangle$

[6] It is 2^{B-1} instead of 2^B because $\bar{\mathbf{y}}$ is in the same orthant of $\mathbf{o}'$.

and $\|\mathbf{y}_{cur}\|$ [7]. v_{max} records the maximum value of $\langle \mathbf{y}_{cur}/\|\mathbf{y}_{cur}\|, \mathbf{o}'\rangle$ during the enumeration. t_{max} records the re-scaling factor which produces v_{max}. The enumeration starts from $t = 0$ (line 1-2). Then, iteratively, we enumerate the next smallest critical value (line 3-4). For every new t, the vector $\mathbf{y}_{cur}$ will change in only one coordinate. Thus, we can update $\mathbf{y}_{cur}$, $\langle \mathbf{y}_{cur}, \mathbf{o}'\rangle$ and $\|\mathbf{y}_{cur}\|$ in $O(1)$ time (line 5). Then we update v_{max} and t_{max} based on the newly enumerated vector (line 6-7). The enumeration terminates when all the critical values have been enumerated (line 3). Finally, we find the optimal vector $\bar{\mathbf{y}}$ via rounding $t_{max} \cdot \mathbf{o}'$ (line 8). We represent $\bar{\mathbf{y}}$ and store it as a vector of unsigned integers $\bar{\mathbf{y}}_u$ (line 9).

The overall time complexity of the algorithm is $O(2^B \cdot D \log D)$ because, in total, we enumerate $(2^{B-1} \cdot D)$ critical values and a min-heap is needed to find the next smallest critical value during the process (its maintenance takes $O(\log D)$ time). We note that this time complexity is good enough for practical usage since our algorithm is designed for vector compression, i.e., B is small. According to our experimental studies in Section 5.2.2, $B = 7$ suffices to stably produce $> 99\%$ recall and $B = 5$ suffices to stably produce $> 95\%$ recall. Under these settings, the quantization of a million-scale dataset of 3,072 dimensions can finish in a few minutes.

Algorithm 1: Quantization

Input : A D-dimensional vector $\mathbf{o}'$; the number of bits per dimension B.
Output: The quantization code $\bar{\mathbf{y}}_u$.

1 $t \leftarrow 0, v_{max} \leftarrow 0, t_{max} \leftarrow 0$
2 Initialize $\mathbf{y}_{cur}$, $\langle \mathbf{y}_{cur}, \mathbf{o}'\rangle$ and $\|\mathbf{y}_{cur}\|$ with $t = 0$
3 **while** *some critical values have not been enumerated* **do**
4 Update t with the next smallest critical value
5 Update $\mathbf{y}_{cur}$, $\langle \mathbf{y}_{cur}, \mathbf{o}'\rangle$ and $\|\mathbf{y}_{cur}\|$ with the new t
6 **if** $\langle \mathbf{y}_{cur}, \mathbf{o}'\rangle / \|\mathbf{y}_{cur}\| > v_{max}$ **then**
7 $v_{max} \leftarrow \langle \mathbf{y}_{cur}, \mathbf{o}'\rangle / \|\mathbf{y}_{cur}\|, t_{max} \leftarrow t$
8 Compute $\bar{\mathbf{y}}$ via rounding $t_{max} \cdot \mathbf{o}'$
9 **return** $\bar{\mathbf{y}}_u$ where $\bar{\mathbf{y}}_u = \bar{\mathbf{y}} + (2^B - 1)/2 \cdot \mathbf{1}$

3.3 Computing the Estimator

Recall that we target to estimate the inner product $\langle \mathbf{o}, \mathbf{q}\rangle$ to further estimate the squared distances (Section 2.2). We adopt the estimator of RaBitQ to inherit its unbiasedness and error bound [8] (see empirical verification in Section 5.2.1), i.e., we use $\langle \bar{\mathbf{o}}, \mathbf{q}\rangle / \langle \bar{\mathbf{o}}, \mathbf{o}\rangle$ to estimate $\langle \mathbf{o}, \mathbf{q}\rangle$. The denominator $\langle \bar{\mathbf{o}}, \mathbf{o}\rangle$ is only related to the data vector and its quantized vector, so it can be pre-computed in the index phase. Thus, we only need to compute $\langle \bar{\mathbf{o}}, \mathbf{q}\rangle$. Recall that $\bar{\mathbf{o}} = P\frac{\bar{\mathbf{y}}}{\|\bar{\mathbf{y}}\|}$. The following equations

[7] For $\mathbf{y}_{cur}$, we use two variables to store $\langle \mathbf{y}_{cur}, \mathbf{o}'\rangle$ and $\|\mathbf{y}_{cur}\|$. Whenever $\mathbf{y}_{cur}$ is updated, we can update these variables to obtain the new values of $\langle \mathbf{y}_{cur}, \mathbf{o}'\rangle$ and $\|\mathbf{y}_{cur}\|$ efficiently.
[8] This conclusion can be directly yielded from the proof in the original RaBitQ paper [27], i.e., Lemma 2.1 holds if (1) the codebook is a set of randomly rotated unit vectors; and (2) $\bar{\mathbf{o}}$ is the nearest vector of $\mathbf{o}$ in the codebook.

illustrate how it can be computed.

$$\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \left\langle P\frac{\bar{\mathbf{y}}}{\|\bar{\mathbf{y}}\|}, \mathbf{q} \right\rangle = \left\langle \frac{\bar{\mathbf{y}}}{\|\bar{\mathbf{y}}\|}, P^{-1}\mathbf{q} \right\rangle = \frac{1}{\|\bar{\mathbf{y}}\|} \langle \bar{\mathbf{y}}, \mathbf{q}' \rangle \tag{11}$$

$$= \frac{1}{\|\bar{\mathbf{y}}\|} \left(\langle \bar{\mathbf{y}}_u, \mathbf{q}' \rangle - \frac{2^B - 1}{2} \sum_{i=1}^{D} \mathbf{q}'[i] \right) \tag{12}$$

Here $\mathbf{q}'$ denotes $P^{-1}\mathbf{q}$; Equation (11) applies an orthonormal matrix P^{-1} to both sides of the inner product; and Equation (12) expresses $\bar{\mathbf{y}}$ with its quantization code $\bar{\mathbf{y}}_u$, i.e., $\bar{\mathbf{y}} = \bar{\mathbf{y}}_u - (2^B - 1)/2 \cdot \mathbf{1}_D$.

Note that $\|\bar{\mathbf{y}}\|$ is only related to the quantized vectors and can be pre-computed in the index phase. $\sum_{i=1}^{D} \mathbf{q}'[i]$ is only related to the query vector. It can be computed once and its time costs can be shared by many data vectors. Thus, the remaining task is the computation of $\langle \bar{\mathbf{y}}_u, \mathbf{q}' \rangle$, i.e., the inner product between a vector of unsigned integers and a vector of floating-point numbers. When $B = 1$ (the case of the original RaBitQ), RaBitQ's implementation can be directly applied [27]. When B equals 4 or 8, the implementations in existing systems (for computing the inner product between a vector of 4-bit or 8-bit unsigned integers and a vector of floating-point numbers) can be directly applied [1, 17]. Other settings of B's can be implemented by splitting a vector of B-bit unsigned integers into several parts, where each part has the size of the power of 2 (e.g., a vector of 9-bit unsigned integers can be split into a binary vector and a vector of 8-bit unsigned integers). We will discuss the details of the idea later in Section 4.1.2.

3.4 Summary and Theoretical Analysis

We summarize the workflow of the extended RaBitQ as follows. In the index phase, the algorithm constructs a quantization codebook by sampling a random orthogonal matrix P. It then applies P^{-1} to the data vectors, normalizes them to obtain $\mathbf{o}'$ and computes their quantization codes via Algorithm 1. In the query phase, when a query comes, it first applies P^{-1} to the query vector and normalizes it to obtain $\mathbf{q}'$. It can then unbiasedly estimate squared distances based on Equation (12) and Equation (2).

Recall that as has been proved in [2] and discussed in Section 3.1, to achieve an error bound ϵ where $\frac{1}{\epsilon^2}\log\frac{1}{\delta} > D$, the minimum required number of bits is $\Theta\left(D\log\left(\frac{1}{D}\cdot\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)\right)$. The following theorem presents that our method achieves this optimality. The proof is left in the technical report [25] due to page limit.

Theorem 3.2. *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 the estimator is bounded by ϵ with probability at least $1-\delta$.*

It is worth noting that the number of bits for each dimension B is *logarithmic* with respect to ϵ^{-2} and is *negatively* related to the dimensionality D. This reveals a counterintuitive fact: with the same number of bits, higher dimensionality results in lower error. Besides the asymptotic analysis, to provide a more quantitative reference, we would also like to present an empirical formula about the error. Note that when $B = 1$ (the original setting of RaBitQ [27]), the error is bounded by $O(1/\sqrt{D})$. When a larger B is used, the error is supposed to decay exponentially. Thus, we present the empirical formula in the following form.

Remark (Empirical Formula). *Let ϵ be the absolute error of estimating inner product of unit vectors. With >99.9% probability, we have $\epsilon < 2^{-B}\cdot c_\epsilon/\sqrt{D}$ where $c_\epsilon = 5.75$.*

We note that the constant in this empirical formula is measured by experimental studies, i.e., we use our algorithm to estimate inner product and collect the statistics of errors (see Section 5.2.7).

Comparison with the Algorithmic Proof in [2]. In the theoretical study [2], there is an algorithmic proof which achieves the asymptotic optimality. However, we note that the algorithmic proof is less practically applicable. Specifically, this proof relies on the operation which represents different integers with different number of bits. Based on this operation, it is proved that the total number of bits needed is asymptotically optimal. This algorithm is unfriendly to real-world systems because it is unclear how a sequence of integers represented by different number of bits can be stored in alignment with each other. In contrast, our method achieves both the asymptotic optimality and the practicality. To the best of our knowledge, our study is also the first which achieves both desiderata at the same time. We would like to emphasize that our work does not target the improvement in terms of theory. Instead, we propose a practically applicable algorithm and prove that it is asymptotically optimal.

4 Applying the Extended RaBitQ to In-memory ANN

4.1 Using the Extended RaBitQ with IVF

4.1.1 The Workflow. Recall that as is discussed in Section 1, the current study targets to compress vectors with a moderate compression rate such that it can avoid storing raw vectors in RAM while still producing good recall (e.g., >90%, >95% or >99%). *Scalar quantization (SQ)* and its variants are the state-of-the-art and also the most popular methods in real-world systems for this need [1, 17, 53, 73]. We note that these methods are usually used together with IVF [17, 53, 73] or graph-based indices [1] for in-memory ANN query. In particular, IVF is a popular method that has been widely deployed in real-world systems for vector search due to its simplicity and effectiveness [17, 53, 73]. It has tiny index size and can be easily combined with various quantization methods due to its sequential memory access pattern in querying.

Specifically, the IVF method partitions the set of data vectors into many clusters (e.g., via KMeans) in the index phase. For every vector in a cluster, we quantize it and store only its quantization code in RAM. Then, in the query phase, when a query comes, it finds a few nearest centroids of the clusters and considers the vectors in these clusters as candidates of ANN. When IVF is used in combination with SQ, it computes an estimated distance for every candidate based on its quantization code and then returns the vector with the smallest estimated distance as the NN. Note that for reducing the memory consumption, when IVF is used with SQ, it does not store the raw vectors in RAM and thus, there is no re-ranking based on the raw vectors. When using the same number of bits for quantization, the extended RaBitQ can produce better accuracy than SQ and its variants (see experimental results in Section 5.2.1). In addition, its computation can be conducted with exactly the same implementations of SQ (Section 3.3). Thus, simply replacing SQ with the extended RaBitQ is expected to yield a better time-accuracy trade-off by improving the accuracy while retaining the efficiency.

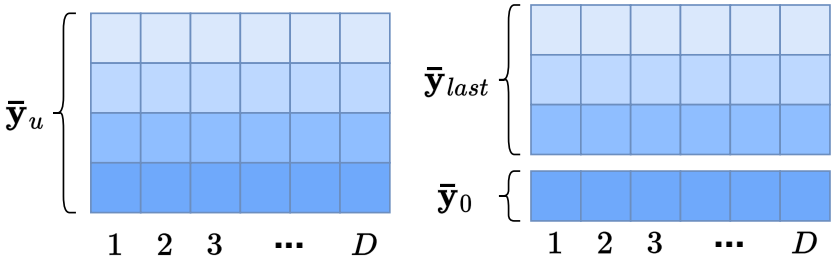


Fig. 2. Decomposition of the Quantization Code $\bar{\mathbf{y}}_u$.

4.1.2 Conducting Distance Comparison with the extended RaBitQ. Beyond simply replacing SQ, we note that the efficiency of the method can be further improved. In particular, a recent study

finds that to reliably find the NN from a set of candidates, it is not necessary to compute an exact distance for every candidate [26]. If an estimated distance can confirm that a candidate is unlikely to be the NN (e.g., a lower bound of the estimated distance is greater than the distance of the NN that has been searched so far), then the candidate can be discarded. Otherwise, it can incrementally compute a more accurate estimated distance until (1) it can confirm that a candidate is unlikely to be the NN; or (2) the exact distance is computed [26]. Based on the idea above, in our case, we note that it is not always necessary to estimate a highly accurate distance with all bits of a quantization code. Instead, we first use a subset of bits of the quantization code to efficiently estimate a distance with lower accuracy. If this estimated distance can confirm that a candidate is unlikely to be the NN, then we discard it. Otherwise, we access the remaining bits to compute a high-accuracy estimated distance based on the full bits of the quantization code.

Specifically, in the quantization code $\bar{\mathbf{y}}_u$, every dimension corresponds to an unsigned integer of B bits. We observe that the concatenation of the most significant bits of all the dimensions of $\bar{\mathbf{y}}_u$ (see Figure 2) exactly equals the quantization code $\bar{\mathbf{x}}_b$ of the original RaBitQ. This is because both of them are decided by the orthant of the vector $\mathbf{o}'$. This motivates us to split a quantization code $\bar{\mathbf{y}}_u$ into two parts, the code of the most significant bits $\bar{\mathbf{y}}_0$ and the code of the remaining bits $\bar{\mathbf{y}}_{last}$, where $\bar{\mathbf{y}}_u = 2^{B-1} \cdot \bar{\mathbf{y}}_0 + \bar{\mathbf{y}}_{last}$ and $\bar{\mathbf{y}}_0 = \bar{\mathbf{x}}_b$. In the query phase, for a set of candidates, we first estimate their distances based on their $\bar{\mathbf{y}}_0$'s, i.e., we compute $\langle \bar{\mathbf{y}}_0, \mathbf{q}' \rangle$ with FastScan [4] as the original RaBitQ does, and further compute the estimated distances based on Lemma 2.1 and Equation (2). Note that this estimated distance is exactly the estimated distance produced by the original RaBitQ and it has a theoretical error bound. Then, we use the bound to decide whether the candidate is unlikely to be the NN. If so, we drop the candidate. Otherwise, we access the code of the remaining bits $\bar{\mathbf{y}}_{last}$ to incrementally compute an estimated distance based on the full bits as follows.

$$\langle \bar{\mathbf{y}}_u, \mathbf{q}' \rangle = \left\langle 2^{B-1} \cdot \bar{\mathbf{y}}_0 + \bar{\mathbf{y}}_{last}, \mathbf{q}' \right\rangle \tag{13}$$

$$= 2^{B-1} \cdot \langle \bar{\mathbf{y}}_0, \mathbf{q}' \rangle + \langle \bar{\mathbf{y}}_{last}, \mathbf{q}' \rangle \tag{14}$$

In particular, as $\langle \bar{\mathbf{y}}_0, \mathbf{q}' \rangle$ has been computed and can be reused, we only need to access $\bar{\mathbf{y}}_{last}$ and compute $\langle \bar{\mathbf{y}}_{last}, \mathbf{q}' \rangle$. When $B = 5$ and $B = 9$, $\bar{\mathbf{y}}_{last}$ corresponds to vectors of 4-bit and 8-bit unsigned integers respectively and the computation of $\langle \bar{\mathbf{y}}_{last}, \mathbf{q}' \rangle$ can be realized with existing implementations [1, 17, 84]. Moreover, we provide the implementations (including designs in both compact storage and efficient computation of inner product) for more settings of B's ($B = 3, 4, 7, 8$) as they also provide valuable trade-off among space, time and accuracy. Due to the limit of space, we leave the details in our code repository. In addition, it is worth noting that this technique only improves the efficiency while hardly affecting the accuracy. Due to the limit of space, we include the ablation study in the technical report [25].

4.2 Using the Extended RaBitQ with HNSW

For graph-based indices, we note that they can be combined with quantization methods for saving memory, i.e., we can replace raw vectors stored for a graph-based index with their quantization codes. During querying, we can estimate distances based on the quantization codes in place of exact distance computation. Due to the limit of space, we present the detailed experimental results in the technical report [25].

5 Experiments

5.1 Experimental Setup

Experimental Platform. All experiments are run on a machine with two Intel Xeon Gold 6418H@4.0GHz CPUs (with Sapphire Rapids architecture, 48 cores/96 threads) and 1TB RAM.

The C++ source codes are compiled by GCC 11.4.0 with `-Ofast -march=native` under Ubuntu 22.04 LTS. For all methods, by following most of the existing studies and benchmarks [6, 23], the search performance is evaluated in a single thread and the indexing time is measured using multiple threads. The source code is available at https://github.com/VectorDB-NTU/Extended-RaBitQ.

Datasets. We evaluate the performance of different algorithms with six million-scale public real-world datasets with varying dimensionality and data types, whose details are presented in Table 1. In addition, we use one dataset with around 100 million vectors of 1,024 dimensions for evaluating the scalability. These datasets encompass both (1) widely adopted benchmarks for evaluating ANN algorithms [6, 27, 44] (such as Word2Vec, MSong and GIST [9]) and (2) embeddings generated by advanced models (including OpenAI-1536 [10], OpenAI-3072 [11], Youtube [12], and MSMARCO [13]). It is worth highlighting that OpenAI-1536 and OpenAI-3072 are produced by the most powerful embedding model *text-embedding-3-large* of OpenAI published in early 2024 [52]. They are generated for evaluating ANN algorithms by Qdrant [61]. As for the set of query vectors, for the datasets which provide the query set themselves, we adopt their query set for testing. For others, we exclude 1,000 vectors from each dataset and use them as the query vectors.

Table 1. Dataset Statistics

Dataset	Size	D	Query Size	Data Type
MSong	992,272	420	200	Audio
Youtube	999,000	1,024	1,000	Video
OpenAI-1536	999,000	1,536	1,000	Text
OpenAI-3072	999,000	3,072	1,000	Text
Word2Vec	1,000,000	300	1,000	Text
GIST	1,000,000	960	1,000	Image
MSMARCO	113,520,750	1,024	1,677	Text

Algorithms. In the experiments regarding the trade-off between the space and the accuracy of distance estimation (Section 5.2.1), we evaluate the performance of the following methods. **(1) RaBitQ+** denotes the extended RaBitQ method proposed in the current study. **(2) RaBitQ (pad)** is the simple extension of RaBitQ mentioned in the original RaBitQ paper which uses more bits by padding the vectors with zeros [27]. **(3) SQ** is the classic uniform scalar quantization method. It has been widely deployed in real-world systems [11, 53, 73, 81]. Specifically, SQ first collects the minimum value v_l and maximum value v_r among all the coordinates of all the vectors. Then, the method uniformly splits the range of values $[v_l, v_r]$ into $2^B - 1$ segments. Each floating point number is then rounded to its nearest boundary of the segments and is represented and stored as a B-bit unsigned integer. **(4) LVQ** is the latest variant of SQ [1]. Different from SQ which collects the smallest and largest values v_l and v_r among all the vectors and performs quantization based on this *global* range of values $[v_l, v_r]$, LVQ collects the v_l and v_r for every *individual* vector and performs quantization of a vector based on its specific range of values $[v_l, v_r]$. **(5) PQ** and **(6) OPQ** are popular quantization methods which are usually used with a large compression rate. They are widely deployed in real-world systems [53, 73, 81]. Note that these methods have two settings $k = 4$ or $k = 8$ (k is the number of bits allocated for quantizing each sub-vector in these methods; see

[9] https://www.cse.cuhk.edu.hk/systems/hash/gqr/datasets.html.
[10] https://huggingface.co/datasets/Qdrant/dbpedia-entities-openai3-text-embedding-3-large-1536-1M
[11] https://huggingface.co/datasets/Qdrant/dbpedia-entities-openai3-text-embedding-3-large-3072-1M
[12] https://research.google.com/youtube8m/download.html
[13] https://huggingface.co/datasets/Cohere/msmarco-v2.1-embed-english-v3

details in their original papers [29, 38]). Since $k = 8$ produces consistently better space-accuracy trade-off than $k = 4$, we report the results of PQ and OPQ with $k = 8$. In addition, a recent method ScaNN [33] proposes a new objective function for training the codebook of PQ. However, it has been reported that the advantage claimed by ScaNN [33] is due to its engineering optimization instead of its algorithm design. With the same compression rates, ScaNN does not deliver better accuracy than PQ, according to the evaluation from the Faiss open-source community [14]. Thus, we exclude ScaNN from the comparison. It is worth noting that when using the same number of bits per dimension, SQ, RaBitQ+ and LVQ have almost the same efficiency in computing inner product and Euclidean distances. However, as has been reported [1, 27], PQ and OPQ have significantly worse efficiency since their computation relies on frequently looking up tables in RAM. Recall that we target to compress vectors such that we do not need to access raw vectors for re-ranking while still producing reasonable recall. Therefore, we focus on evaluating these methods in a moderate compression rate, i.e., the number of bits per dimension ranges from 1 to 10. When a larger compression rate is used, none of methods can stably produce over 90% recall without re-ranking. When a smaller compression rate is used, it would be wasteful since 10 bits per dimension suffice to produce nearly perfect recall. In this experiment, for the sake of fairness, as a pre-procession for all the methods, we center the datasets with their global centroid [15]. Note that centering is a common operation for boosting the accuracy and is mentioned by many quantization algorithms including PQ and LVQ in their original papers [1, 38] (a.k.a., quantizing the residual vectors). It is also applied to other quantization algorithms (e.g., OPQ and SQ) in the implementation of Faiss' code repository. In the experiments regarding the ANN query (Section 5.2.2), based on the experimental results in Section 5.2.1, we compare RaBitQ+ with the most competitive baseline LVQ [1]. We combine RaBitQ+ and LVQ with the IVF index as is discussed in Section 4. In this experiment, as a pre-procession for all the methods, we center every cluster in the IVF index with its local centroid. All the methods are optimized with the SIMD instructions till AVX512.

Performance Metrics. In the experiments regarding the trade-off between the space and the accuracy of distance estimation (Section 5.2.1), we measure the accuracy with (1) the *average relative error* and (2) the *maximum relative error* on the estimated squared distances. We measure the space with the number of bits per dimension. Specifically, it sums up all the space consumption of a vector and divides it by the dimensionality. Note that for RaBitQ+, it also covers the space for storing two floating-point numbers for every vector, i.e., $\|\mathbf{o}_r - \mathbf{c}\|$ and $1/(\|\mathbf{y}_u\| \cdot \langle \mathbf{o}, \bar{\mathbf{o}} \rangle)$. In the experiments regarding the ANN query (Section 5.2.2), we adopt *recall* and *average distance ratio* for measuring the accuracy of ANN search. *Recall* is the percentage of successfully retrieved true nearest neighbors. *Average distance ratio* is the average of the distance ratios of the retrieved nearest neighbors over the true nearest neighbors. These metrics are widely adopted to measure the accuracy of ANN algorithms [6, 24, 34, 44, 57, 67]. We adopt *query per second (QPS)*, i.e., the number of queries a method can handle in a second, to measure the efficiency. It is widely adopted to measure the efficiency of ANN algorithms [6, 44, 76]. Following [6, 44, 76], the *query time* is evaluated in a single thread and the search is conducted for each query individually (instead of queries in a batch). All the metrics are measured on every single query and averaged over the whole query set. We also report the time costs of different methods in the index phase.

Parameter Settings in ANN Query. In the IVF index, we set the number of clusters to be 4,096 for million-scale datasets by following the suggestions of Faiss [20]. For the MSMARCO dataset, we use 262,144 clusters. For RaBitQ+, we implement our algorithm with $B = 3, 4, 5, 7, 8, 9$ and conduct ANN query with the strategy presented in Section 4. For the alignment of data, we pad the vectors

[14] https://github.com/facebookresearch/faiss/wiki/Indexing-1M-vectors#4-bit-pq-comparison-with-scann

[15] With the centroid $\mathbf{c}$, the centering operation is to replace every data and query vector $\mathbf{a}$ with $\mathbf{a} - \mathbf{c}$.

with zeros such that their dimensionality is the minimum multiple of 64 which is larger than their dimensionality (i.e., we pad the dimensionality of MSong to 448 and that of Word2Vec to 320). For LVQ, we apply the settings $B = 4$ and $B = 8$ in its original paper [1].

5.2 Experimental Results

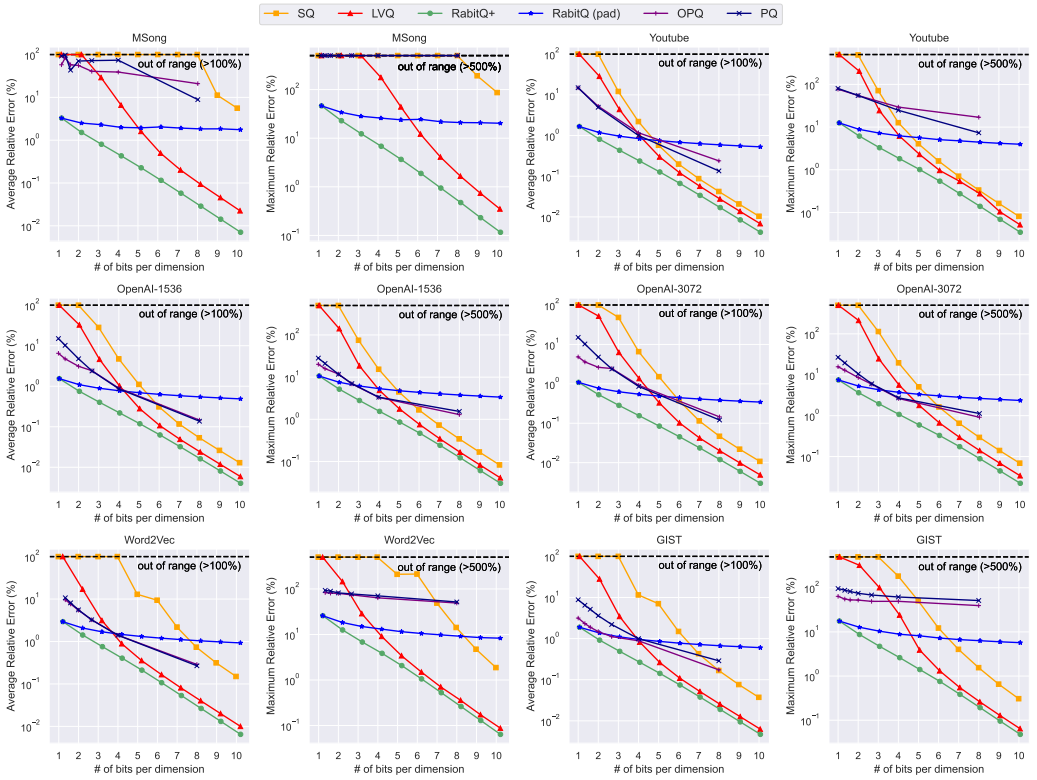


Fig. 3. Space-Accuracy Trade-Off for Distance Estimation (Log-Scale).

5.2.1 Space-Accuracy Trade-Off for Distance Estimation. In this experiment, we evaluate different quantization methods by using them for estimating the distances between data vectors and query vectors. For each method, in Figure 3, we vary their number of bits and plot the curves of the average relative error (left panels, lower left is better) and maximum relative error (right panels, lower left is better) to investigate their space-accuracy trade-off. We, in particular, focus on the number of bits from $B = 1$ to $B = 10$ which correspond to a moderate compression rate. Note that $B = 7$ suffices to produce $> 99\%$ recall (Section 5.2.2).

According to Figure 3, RaBitQ+ (the green curve) stably achieves better accuracy than all the baseline methods on all the tested datasets when using the same number of bits. Specifically, we have the following observations. (1) RaBitQ+ v.s. SQ and LVQ: We observe that when $B > 6$, under the same number of bits, the average relative error of LVQ is consistently larger than ours by 1.3x-3.1x (the gap of SQ's error from ours is even larger). When $B < 6$, the gap is even larger. When $B = 1$ or $B = 2$, LVQ and SQ hardly produce reasonable accuracy and their errors are larger than ours by orders of magnitude. The primary improvement of RaBitQ+ is most notable when the number of bits is small, which we believe is critical for advancing the effectiveness of quantization in ANN query. It is worth highlighting that 8-bit quantization already achieves nearly perfect recall

for most ANN queries (as is shown in Section 5.2.2), indicating that further error reductions at this compression rate and below are less impactful. In contrast, the challenge lies in maintaining high accuracy while using fewer bits for higher compression rates. Note that for RaBitQ+, LVQ and SQ, the computation of distances or inner product can be implemented in almost identical ways (Section 3.3). This result implies that simply replacing SQ and LVQ in existing systems with RaBitQ+ would stably improve the performance. Moreover, when $B = 1$, the improvement would be especially significant. As a comparison, the estimated distances are much less accurate when LVQ uses 1 bit only. As shown in Figure 3, its error is larger than that of RaBitQ+ by several orders of magnitude and is hardly practical (e.g., the relative error exceeds 100%). This unique advantage enables RaBitQ+ to compute a fairly accurate distance based on the first bit of the quantization codes and prune many candidates, which improves the efficiency. It will be reflected later in Section 5.2.2. (2) PQ and OPQ: We find that PQ and OPQ have reasonable accuracy in most datasets when the number of bits per dimension is small (e.g., 1 or 2). However, when the number of bits increases, the errors of these methods do not decay as fast as RaBitQ+, SQ or LVQ. In particular, PQ and OPQ fail to outperform SQ and LVQ usually when the number of bits per dimension is ≥ 4. Similar results have also been observed in the LVQ paper [1]. Moreover, it has also been reported that the efficiency of PQ and OPQ is significantly worse than SQ and LVQ because for computing distances or inner product, they rely on looking up tables in RAM [1, 3]. When FastScan is applied to accelerate the computation (which requires to set $k = 4$), the space-accuracy trade-off would be even worse [4]. These results indicate that PQ and OPQ can hardly be competitive in compressing high-dimensional vectors with moderate compression rates. (3) RaBitQ (pad): We find that RaBitQ (pad) is competitive when the number of bits per dimension is small (e.g., 1 or 2). However, the error of RaBitQ (pad) also decays slowly. This is because as has been theoretically proved in [27], its error decays in a trend of $O(1/\sqrt{B \cdot D})$. As a comparison, the errors of RaBitQ+, SQ and LVQ decay in an exponential trend with respect to B.

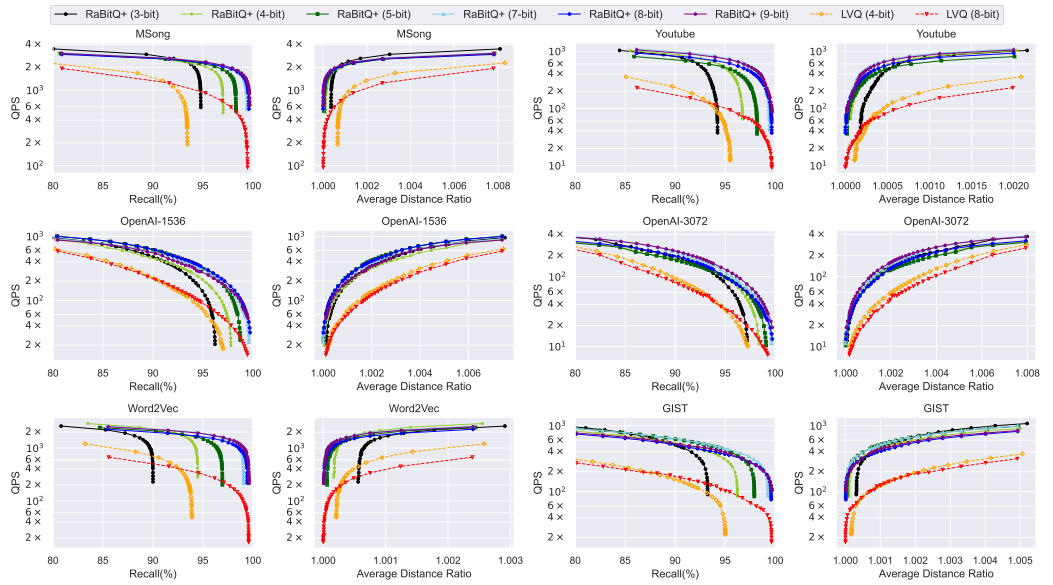


Fig. 4. Time-Accuracy Trade-Off for the ANN Query (Log-Scale), $K = 100$. All the methods are combined with the IVF index.

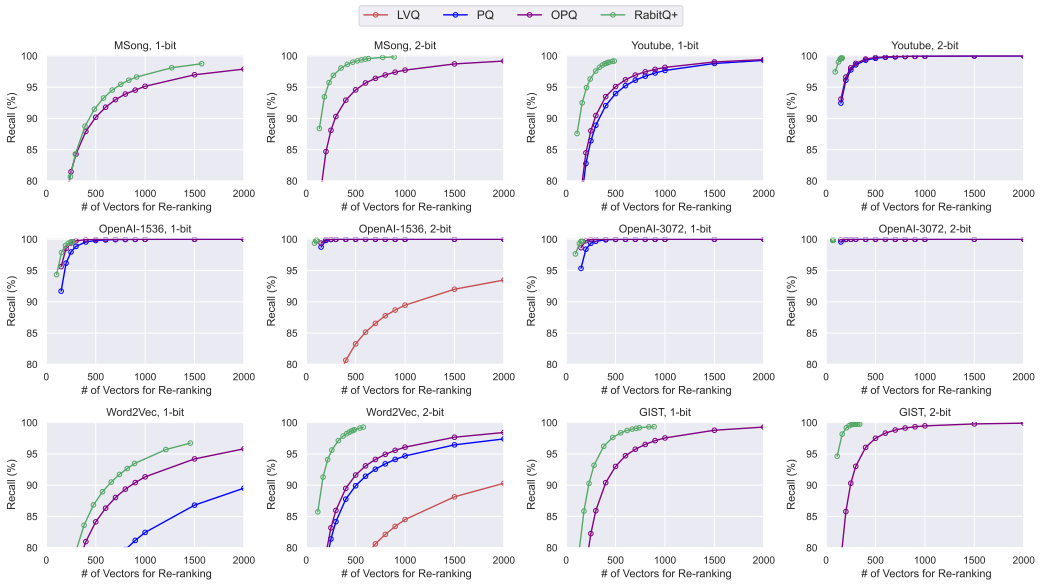


Fig. 5. Recall-Re-Ranking Trade-Off for the ANN Query (Log-Scale), $K = 100$. The missing curve of a certain method indicates that it fails to achieve 80% recall on the dataset with re-ranking.

5.2.2 Time-Accuracy Trade-Off for ANN Query. In this experiment, we evaluate different quantization methods by using them in combination with the IVF index for ANN queries. In Figure 4, we plot for the methods their curves of "QPS"-"Recall" (left panels, upper right is better) and curves of "QPS"-"Average Distance Ratio" (right panels, upper left is better) by varying the number of clusters to probe to investigate their time-accuracy trade-off.

According to Figure 4, we have the following observations. (1) Time-accuracy trade-off: We observe that with the same number of bits (4-bit and 8-bit), RaBitQ+ outperforms LVQ in terms of time-accuracy trade-off. It is worth highlighting that the improvement in efficiency under the same recall is significant since RaBitQ+ can prune many candidates with the estimated distances based on the first bits of the quantization codes. In addition, the estimation can be implemented with FastScan [4] which is highly efficient (Section 4.1.2). (2) Recall: We find that 4-bit, 5-bit and 7-bit quantization suffices to produce over 90%, 95% and 99% recall on all the tested datasets respectively without accessing the raw vectors for re-ranking. (3) Average distance ratio: We observe that the average distance ratio produced by 5-bit quantization of RaBitQ+ is nearly perfect across all the datasets although it cannot achieve perfect recall (i.e., it usually produces 97% recall only). This indicates that the retrieved data vectors have their distances from the query extremely close to those of the true NNs. (4) Results on datasets from OpenAI: We find that the ANN algorithm with RaBitQ+ on these datasets is highly robust, e.g., 3-bit quantization suffices to produce $> 95\%$ recall.

5.2.3 Space and Time Costs of Indexing. In this section, we report the space and time costs of indexing. In Table 2, we separately report the space/time costs of the IVF index and those of the quantization algorithms. We have the following observations. (1) The time costs of quantizing the vectors via RaBitQ+ and LVQ are significantly smaller than that of building the IVF index. Thus, their time costs of quantization are not a bottleneck. However, PQ and OPQ have their time costs comparable or larger than that of IVF, indicating that their overhead is unignorable. (2) The space

Table 2. Space (GB) / Indexing Time (mins) costs for different methods. We separately report the space/time costs of the IVF index and those of the quantization algorithms. As a reference, we also present the space costs of the raw datasets. The missing parts in the table indicate that the algorithm fails on this dataset.

	Raw	IVF	RaBitQ+		LVQ		PQ		OPQ	
Num. of bits			4	8	4	8	4	8	4	8
MSong	1.56/-	0.01/0.6	0.22/<0.1	0.43/0.2	0.21/<0.1	0.40/<0.1	0.19/0.6	0.39/1.0	0.19/3.4	0.39/7.0
Youtube	3.82/-	0.02/1.3	0.50/<0.1	0.97/0.6	0.49/<0.1	0.96/<0.1	0.48/1.3	0.95/2.7	0.48/7.5	0.95/15
OpenAI-1536	5.72/-	0.02/2.1	0.75/0.1	1.46/0.8	0.73/<0.1	1.45/<0.1	0.71/2.0	1.43/3.7	0.71/8.5	1.43/17
OpenAI-3072	11.44/-	0.05/5.0	1.49/1.0	2.92/2.3	1.44/<0.1	2.87/<0.1	1.43/4.0	2.86/7.7	1.43/25	2.86/47
Word2Vec	1.12/-	0.01/0.5	0.16/<0.1	0.31/0.2	0.15/<0.1	0.29/<0.1	0.14/0.4	0.28/0.7	0.14/2.3	0.28/5.1
GIST	3.58/-	0.01/1.3	0.47/<0.1	0.92/0.4	0.47/<0.1	0.91/<0.1	0.45/1.2	0.89/2.5	0.45/7.5	0.89/15
MSMARCO	433.47/-	1.0/>1d	56/10.6	110/58.6	56/0.8	110/1.3	-/-	-/-	-/-	-/-

cost of IVF is almost ignorable, because in IVF, one cluster hosts hundreds of vectors and its space consumption can be largely amortized.

5.2.4 Recall-Re-Ranking Tradeoff for ANN Query. To provide more comprehensive evaluation and enable extrapolation to disk-based configurations, in this section, we extend the evaluation to the trade-off between the number of raw vectors accessed for re-ranking and the recall. Note that the number of vectors for re-ranking indicates the I/O cost. Based on the experimental results in Section 5.2.1, we only include RaBitQ+, LVQ, PQ and OPQ in the evaluation. Because with a moderate compression rate applied (>2-bit), a method can produce good recall without re-ranking, we focus on evaluating the methods with large compression rates (i.e., 1-bit and 2-bit).

In Figure 5, we plot the the curve of the trade-off between recall and the number of raw vectors accessed for re-ranking. Specifically, for all methods, we vary a parameter which controls the number of candidates for re-ranking. For RaBitQ+, we would like to highlight that, compared with the existing quantization methods, in the scenario with re-ranking, RaBitQ+ has an unique advantage due to its rigorous theoretical error bound. Specifically, based on the bound, RaBitQ+ can skip the re-ranking of some vectors, i.e., if the lower bound of a vector's distance is larger than the upper bound of the current NN's distance, then we can skip re-ranking the vector. This unique advantage reduces the number of vectors for re-ranking in several datasets while preserving high recall with theoretical guarantees. We have the following observations. (1) Overall, RaBitQ+ achieves dominant performance in the trade-off between recall and the number of vectors for re-ranking over all baselines. In particular, LVQ, the state-of-the-art variant of scalar quantization, fails to achieve reasonable recall even with re-ranking applied when $B = 1$ and $B = 2$. In addition, with re-ranking applied, the baselines PQ and OPQ have shown competitive accuracy. However, as has been discussed [1], the efficiency of PQ and OPQ is significantly worse than RaBitQ+, LVQ and SQ. (2) For the datasets, OpenAI-1536 and OpenAI-3072, RaBitQ+, PQ and OPQ can produce promising accuracy via re-ranking only a small number of candidates while LVQ fails to produce reasonable accuracy. In particular, on OpenAI-3072, when 2 bits are used, RaBitQ+ can achieve nearly perfect recall via re-ranking *fewer than* K vectors. However, all other methods need re-rank $> K$ candidates to boost the accuracy. To the best of our knowledge, our method is the first that achieves the result.

5.2.5 Verifying the Scalability. In this section, we study the scalability of RaBitQ+ on the MSMARCO dataset with about 100 millions of 1,024-dimensional data vectors. We report that the quantization of the dataset with $B = 9$ can be finished in around 2 hours, which is not a bottleneck (as a comparison, building the IVF index for the dataset takes more than 1 day). Figure 6 reports the time-accuracy trade-off for ANN queries. It shows that RaBitQ+ still achieves consistently better time-accuracy trade-off compared with LVQ. In particular, with RaBitQ+, using $B = 4$ suffices to produce >95%

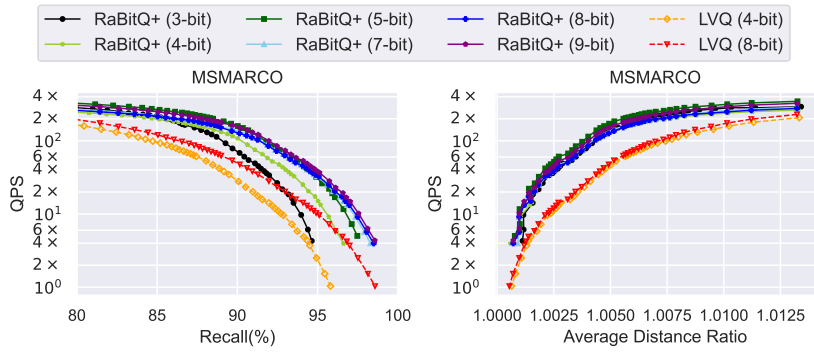


Fig. 6. Verification Study for Scalability.

recall without re-ranking. In this case, the space consumption of RaBitQ+ is 56.94 GB while the raw dataset takes 433.47 GB.

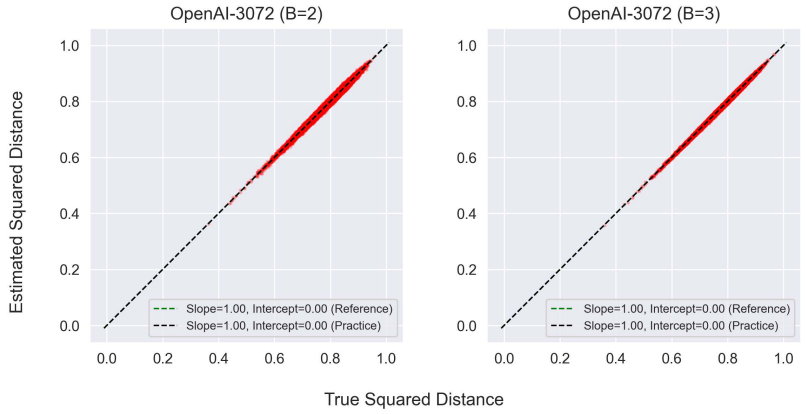


Fig. 7. Verification Study for Unbiasedness.

5.2.6 Verifying the Unbiasedness. In this section, we verify that based on our new codebook with more vectors than those of the original RaBitQ method, the estimator is still unbiased. Due to the limit of space, we only present the results for $B = 2, 3$ in two panels of Figure 7 respectively. In each panel, we collect around 10^7 pairs of the estimated squared distances and the true squared distances (between the first 10 query vectors and about 10^6 data vectors in the OpenAI-3072 dataset) which are normalized with the maximum true squared distances. To verify the unbiasedness, we fit these pairs with linear regression and plot the result with the black dashed line. Note that in all panels, this line has the slope of 1 and the y-axis intercept of 0. This indicates that RaBitQ+ provides unbiased distance estimation.

5.2.7 Measuring the Empirical Formula among ϵ, D and B. In this section, we measure the constants in the empirical formula which presents the relationship among the absolute error of estimating inner product between unit vectors, B and D (see Section 3.4). In particular, for a pair of B and D, we randomly sample 5×10^6 pairs of unit vectors (i.e., each vector is sampled from standard Gaussian distribution and is normalized then), apply RaBitQ+ to estimate their inner product and collect the error of the estimation. In the left panel of Figure 8, we fix $D = 1000$ and plot the curve of the 99.9% quantile of the empirical errors with respect to B (the red curve). In the right panel, on

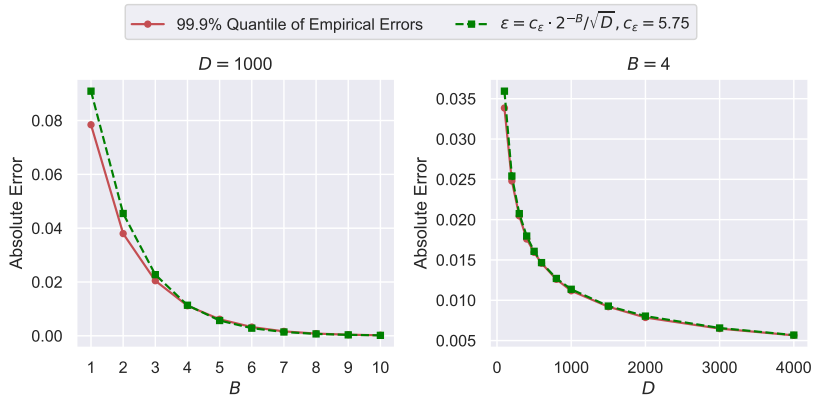


Fig. 8. Measure the Constants in the Empirical Formula.

the other hand, we fix $B = 4$ and plot the curve of the 99.9% quantile of the empirical errors with respect to D (the red curve). Note that our empirical formula is in the form of $\epsilon < c_\epsilon \cdot 2^{-B}/\sqrt{D}$ (see Section 3.4). We measure the constant c_ϵ by tuning c_ϵ such that the curve of $\epsilon = c_\epsilon \cdot 2^{-B}/\sqrt{D}$ (the green curve) is higher than the curve of the 99.9% quantile of the empirical errors. The result shows that when $c_\epsilon = 5.75$, the curves match well.

6 Related Work

Quantization. A substantial body of literature on the quantization of high-dimensional vectors exists across various fields, including machine learning, computer vision, and data management [1, 8, 21, 29, 30, 38, 39, 47, 55, 71, 71, 79, 83]. We refer readers to comprehensive surveys and textbooks [19, 48, 63, 72, 74, 82]. The existing studies about quantization can be divided into roughly two threads: (1) PQ and its variant [8, 29, 38, 47, 55] and (2) SQ and its variants [1, 21, 79]. We note that although the family of PQ and the family of SQ have highly diversified schemes, they can be presented in a unified framework: (1) in the index phase, they construct a quantization codebook and for each data vector, they find the nearest vector in it as the quantized data vector; and (2) in the query phase, they estimate the distance between a data vector and a query vector based on the quantized data vector. Despite this, we observe that in existing literature [17, 29, 38, 46, 48], PQ and SQ were seldom compared to each other [16]. Indeed, PQ and its variants are used mostly in the scenarios with large compression rates [17, 29, 38, 46, 48, 72, 74, 83] while SQ and its variants are used mostly in the scenarios with moderate compression rates [1, 17, 73, 81]. According to our experimental results in Section 5.2.1, we find that with moderate compression rates (e.g., quantizing the vectors with ≥ 4 bits per dimension), PQ does not outperform the classical SQ on many datasets. In addition, as has been reported [1], PQ is significantly slower than SQ in distance computation when the same number of bits are used. On the other hand, with relatively large compression rates (e.g., quantizing the vectors with 1 or 2 bits per dimension), SQ can hardly produce reasonable accuracy. This may help explain why PQ and SQ were usually regarded as two separate lines of studies, i.e., each of them is capable of a certain scenario only. In contrast, our extended RaBitQ method excels in both scenarios. With the compression rates from 3x to 32x, our method stably outperforms all the methods empirically (Section 5.2.1). Moreover, it achieves the asymptotic optimality in theory (Section 3.4). In addition, it is worth noting that PQ and its variants are also used with a compression rate which is even larger than 32x. However, in this case, without re-ranking, they can

[16]The LVQ paper [1] compares PQ and SQ and finds that PQ does not have competitive performance with a moderate compression rate.

only produce poor recall [17, 29, 38, 46, 48, 72, 74, 83] (e.g., <80%), which deviates from the target of this study, i.e., achieving high recall without storing the raw vectors in RAM for re-ranking. For better comprehensiveness, we include the discussion on the way of using our method with a larger compression rate in Section 7 and leave more detailed study as future work.

ANN Query. ANN query is a key component of high-dimensional vector data management and has been widely supported by real-world systems [32, 50, 53, 73, 81]. Among the existing algorithms [12, 15, 23, 29, 38, 44, 45, 78, 80], the partition-based method IVF and the graph-based method HNSW have been deployed to the widest extent [45, 53, 73, 81]. For more methods, we refer readers to recent tutorials [18, 54, 62], surveys [7, 53, 58, 77] and benchmarks [6, 16, 44, 76]. Besides the ANN query, there have been growing interests in many advanced queries related to ANN [50, 53, 73]. For example, in real-world systems, besides vectors, a data object usually involves several attributes such as labels, numbers and strings. It is often the case that users target to find the data objects which satisfy some constraints on the attributes and have their vector nearest to the user's query vector. These questions are usually referred to as attribute-filtering ANN query [50, 53, 56, 73, 81, 86]. Besides, due to the recent development in information retrieval, there has also been a trend in multi-vector search [41, 53, 64]. In particular, these studies map a document to multiple high-dimensional vectors. During querying, they also map a query document to multiple vectors and search for the most relevant documents via a new similarity metric called *MaxSim*, which is aggregated from the inner products between query and data vectors [41, 64]. We note that in this paper, we present a method of vector quantization and support to estimate inner product and squared Euclidean distances unbiasedly based on the quantized vectors. Our method can be used in these tasks for vector quantization and distances/inner-product estimation seamlessly.

Random Projection. Random projection is a fundamental technique that has wide applications [10, 14, 15, 26, 27, 34, 67, 68]. The effectiveness of random projection is closely related to the seminal Johnson-Lindenstrauss (JL) Lemma [40]. In particular, JL Lemma states that projecting a vector onto the space of $O(\epsilon^{-2}\log(1/\delta))$ dimensions suffices to provide an error bound of ϵ with the probability of at least $1-\delta$ [17]. A theoretical study proves that JL Lemma is asymptotically optimal in terms of the trade-off between the number of *dimensions* and the error bound [43]. For more applications and theoretical conclusions, we refer readers to a comprehensive introduction [22]. Moreover, several studies further investigate on the trade-off between the number of *bits* for representing a vector and the error bound of inner product/distance estimation [2, 36, 42]. In particular, they prove that compressing a vector into a short code with $O(\epsilon^{-2}\log(1/\delta))$ bits suffices to guarantee an error bound of ϵ. As a comparison, trivially applying JL Lemma and quantizing the real value of each dimension with SQ requires $O(\log\frac{1}{\epsilon})$ bits per dimension [2]. Furthermore, it is observed in [2], for the first time, that when the required accuracy is high (ϵ is small, $\epsilon^{-2}\log(1/\delta) > D$), the aforementioned result can be further improved (see Section 3.1). However, as has been discussed in Section 3.4, the algorithmic proof in this study is hardly practically applicable.

7 Conclusion

In conclusion, in this study, we extend RaBitQ for quantization with multiple bits per dimension. It supports to compress high-dimensional vectors with a moderate compression rate such that it produces promising recall without re-ranking. The method has consistent advantages in both empirical accuracy and theoretical guarantees over existing quantization methods including PQ, SQ and their variants. In addition, it supports efficient distance comparisons by first estimating a distance based on the first bits of the quantization codes, which helps prune many candidates from

[17] The error bound includes a multiplicative error bound of Euclidean distances and an additive error bound of inner product.

the distance computation based on the full codes. Extensive experiments verify its effectiveness in practice and the alignment between the empirical performance and the theoretical analysis.

We would like to discuss on the following applications and extensions. (1) Like the original RaBitQ, the extended RaBitQ method can be adapted to support the estimation of inner product and cosine similarity (see [27]). (2) When targeting to compress vectors with a compression rate of >32x (i.e., we use fewer than 1 bit per dimension for quantizing a vector), we could first apply random projection for dimension reduction and apply the original RaBitQ method for binarization. The details are left in the technical report [25] due to the page limit. (3) Our method may bring benefits to other scenarios of ANN search (e.g., we may put the first bits of the quantization codes in RAM, leave the remaining bits in SSDs and conduct distance comparisons like what we do in Section 4.1.2). (4) Quantization is also a fundamental component in machine learning systems. The practicality and optimality of the extended RaBitQ method implies its potential of optimizing neural network inferencing and training.

Acknowledgements

We would like to thank Songyuan Wu for valuable discussions and the anonymous reviewers for providing constructive feedback and valuable suggestions. This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Award MOE-T2EP20221-0013 and Tier 1 Awards (RG20/24 and RG77/21). This work of Raymond Chi-Wing Wong was supported in part by WEB24EG01-A. 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

[1] Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore Willke. 2023. Similarity Search in the Blink of an Eye with Compressed Indices. *Proc. VLDB Endow.* 16, 11 (aug 2023), 3433–3446. doi:10.14778/3611479.3611537

[2] Noga Alon and Bo'az Klartag. 2017. Optimal Compression of Approximate Inner Products and Dimension Reduction. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. 639–650. doi:10.1109/FOCS.2017.65

[3] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2015. Cache Locality is Not Enough: High-Performance Nearest Neighbor Search with Product Quantization Fast Scan. *Proc. VLDB Endow.* 9, 4 (dec 2015), 288–299. doi:10.14778/2856318.2856324

[4] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2017. Accelerated Nearest Neighbor Search with Quick ADC. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval* (Bucharest, Romania) *(ICMR '17)*. Association for Computing Machinery, New York, NY, USA, 159–166. doi:10.1145/3078971.3078992

[5] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2021. Quicker ADC : Unlocking the Hidden Potential of Product Quantization With SIMD. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 5 (2021), 1666–1677. doi:10.1109/TPAMI.2019.2952606

[6] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. *Inf. Syst.* 87, C (jan 2020), 13 pages. doi:10.1016/j.is.2019.02.006

[7] Martin Aumüller and Matteo Ceccarello. 2023. Recent Approaches and Trends in Approximate Nearest Neighbor Search, with Remarks on Benchmarking. *Data Engineering* (2023), 89.

[8] Artem Babenko and Victor Lempitsky. 2014. Additive Quantization for Extreme Vector Compression. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 931–938. doi:10.1109/CVPR.2014.124

[9] Object Box. 2024. Object Box. https://objectbox.io/vector-database-for-ondevice-ai/.

[10] Moses S. Charikar. 2002. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing* (Montreal, Quebec, Canada) *(STOC '02)*. Association for Computing Machinery, New York, NY, USA, 380–388. doi:10.1145/509907.509965

[11] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*.

[12] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 426–435.

[13] Couchbase. 2024. Vector Search at the Edge with Couchbase Mobile. https://www.couchbase.com/blog/vector-search-at-the-edge-with-couchbase-mobile/.

[14] Sanjoy Dasgupta and Yoav Freund. 2008. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 537–546.

[15] 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.

[16] Magdalen Dobson, Zheqi Shen, Guy E Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. 2023. Scaling Graph-Based ANNS Algorithms to Billion-Size Datasets: A Comparative Analysis. *arXiv preprint arXiv:2305.04359* (2023).

[17] 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:2401.08281 [cs.LG] https://arxiv.org/abs/2401.08281

[18] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. New Trends in High-D Vector Similarity Search: Al-Driven, Progressive, and Distributed. *Proc. VLDB Endow.* 14, 12 (jul 2021), 3198–3201. doi:10.14778/3476311.3476407

[19] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2018. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *Proc. VLDB Endow.* 12, 2 (oct 2018), 112–127. doi:10.14778/3282495.3282498

[20] Faiss. 2025. Faiss. https://github.com/facebookresearch/faiss.

[21] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2000. Vector Approximation Based Indexing for Non-Uniform High Dimensional Data Sets. In *Proceedings of the Ninth International Conference on Information and Knowledge Management* (McLean, Virginia, USA) *(CIKM '00)*. Association for Computing Machinery, New York, NY, USA, 202–209. doi:10.1145/354756.354820

[22] Casper Benjamin Freksen. 2021. An Introduction to Johnson-Lindenstrauss Transforms. *CoRR* abs/2103.00564 (2021). arXiv:2103.00564 https://arxiv.org/abs/2103.00564

[23] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (jan 2019), 461–474. doi:10.14778/3303753.3303754

[24] 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* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 541–552. doi:10.1145/2213836.2213898

[25] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2024. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search (Technical Report). https://github.com/VectorDB-NTU/Extended-RaBitQ/blob/main/technical_report.pdf.

[26] Jianyang Gao and Cheng Long. 2023. High-Dimensional Approximate Nearest Neighbor Search: With Reliable and Efficient Distance Comparison Operations. *Proc. ACM Manag. Data* 1, 2, Article 137 (jun 2023), 27 pages. doi:10.1145/3589282

[27] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 3, Article 167 (may 2024), 27 pages. doi:10.1145/3654970

[28] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[29] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2946–2953.

[30] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 12 (2013), 2916–2929. doi:10.1109/TPAMI.2012.193

[31] Ant Group. 2024. VSAG. https://github.com/antgroup/vsag.

[32] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: A Cloud Native Vector Database Management System. *Proc. VLDB Endow.* 15, 12 (aug 2022), 3548–3561. doi:10.14778/3554821.3554843

[33] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *Proceedings of the 37th International Conference on*

Machine Learning (ICML'20). JMLR.org, Article 364, 10 pages.

[34] 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.

[35] 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.

[36] Piotr Indyk and Tal Wagner. 2022. Optimal (Euclidean) Metric Compression. *SIAM J. Comput.* 51, 3 (2022), 467–491. doi:10.1137/20M1371324 arXiv:https://doi.org/10.1137/20M1371324

[37] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf

[38] 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.

[39] Wenqi Jiang, Shigang Li, Yu Zhu, Johannes De Fine Licht, Zhenhao He, Runbin Shi, Cedric Renggli, Shuai Zhang, Theodoros Rekatsinas, Torsten Hoefler, and Gustavo Alonso. 2023. Co-design Hardware and Algorithm for Vector Search. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 87, 15 pages. doi:10.1145/3581784.3607045

[40] William B Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space 26. *Contemporary mathematics* 26 (1984), 28.

[41] 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.

[42] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. 1998. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA) *(STOC '98)*. Association for Computing Machinery, New York, NY, USA, 614–623. doi:10.1145/276698.276877

[43] Kasper Green Larsen and Jelani Nelson. 2017. Optimality of the Johnson-Lindenstrauss lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 633–638.

[44] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.

[45] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836. doi:10.1109/TPAMI.2018.2889473

[46] Julieta Martinez, Joris Clement, Holger H. Hoos, and James J. Little. 2016. Revisiting Additive Quantization. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 137–153.

[47] Julieta Martinez, Shobhit Zakhmi, Holger H. Hoos, and James J. Little. 2018. LSQ++: Lower Running Time and Higher Recall in Multi-Codebook Quantization. In *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI* (Munich, Germany). Springer-Verlag, Berlin, Heidelberg, 508–523. doi:10.1007/978-3-030-01270-0_30

[48] 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.

[49] Milvus. 2024. Run Milvus Lite Locally. https://milvus.io/docs/install-overview.md#Milvus-Lite.

[50] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F. Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. 2023. High-Throughput Vector Similarity Search in Knowledge Graphs. *Proc. ACM Manag. Data* 1, 2, Article 197 (jun 2023), 25 pages. doi:10.1145/3589777

[51] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence* 36, 11 (2014), 2227–2240.

[52] OpenAI. 2024. New embedding models and API updates. https://openai.com/index/new-embedding-models-and-api-updates/.

[53] James Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *The VLDB Journal* (07 2024), 1–25. doi:10.1007/s00778-024-00864-x

[54] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Vector Database Management Techniques and Systems. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) *(SIGMOD/PODS '24)*. Association for Computing Machinery, New York, NY, USA, 597–604. doi:10.1145/3626246.3654691

[55] John Paparrizos, Ikraduya Edian, Chunwei Liu, Aaron J. Elmore, and Michael J. Franklin. 2022. Fast Adaptive Similarity Search through Variance-Aware Quantization. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 2969–2983. doi:10.1109/ICDE53745.2022.00268

[56] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proc. ACM Manag. Data* 2, 3, Article 120 (may 2024), 27 pages. doi:10.1145/3654923

[57] Marco Patella and Paolo Ciaccia. 2008. The Many Facets of Approximate Similarity Search. In *First International Workshop on Similarity Search and Applications (sisap 2008)*. 10–21. doi:10.1109/SISAP.2008.18

[58] Marco Patella and Paolo Ciaccia. 2009. Approximate Similarity Search: A Multi-Faceted Problem. *J. of Discrete Algorithms* 7, 1 (mar 2009), 36–48. doi:10.1016/j.jda.2008.09.014

[59] pgvector. 2024. pgvector. https://github.com/pgvector/pgvector.

[60] pgvector.rs. 2024. pgvector.rs. https://pgvecto.rs/.

[61] Qdrant. 2024. Qdrant. https://qdrant.tech/.

[62] Jianbin Qin, Wei Wang, Chuan Xiao, Ying Zhang, and Yaoshu Wang. 2021. High-Dimensional Similarity Query Processing for Data Science. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 4062–4063. doi:10.1145/3447548.3470811

[63] Hanan Samet. 2005. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[64] 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).

[65] SingleStore. 2024. SingleStore. https://www.singlestore.com/built-in-vector-database/.

[66] Yongye Su, Yinqi Sun, Minjia Zhang, and Jianguo Wang. 2024. Vexless: A Serverless Vector Data Management System Using Cloud Functions. *Proc. ACM Manag. Data* 2, 3, Article 187 (may 2024), 26 pages. doi:10.1145/3654990

[67] 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).

[68] 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.

[69] TensorChord. 2024. VectorChord. https://github.com/tensorchord/VectorChord.

[70] Benjamin Trent. 2025. Better Binary Quantization (BBQ) in Lucene and Elasticsearch. https://www.elastic.co/search-labs/blog/better-binary-quantization-lucene-elasticsearch.

[71] Ertem Tuncel, Hakan Ferhatosmanoglu, and Kenneth Rose. 2002. VQ-Index: An Index Structure for Similarity Searching in Multimedia Databases. In *Proceedings of the Tenth ACM International Conference on Multimedia* (Juan-les-Pins, France) *(MULTIMEDIA '02)*. Association for Computing Machinery, New York, NY, USA, 543–552. doi:10.1145/641007.641117

[72] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2016. Learning to Hash for Indexing Big Data - A Survey. *Proc. IEEE* 104, 1 (2016), 34–57. doi:10.1109/JPROC.2015.2487976

[73] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2614–2627. doi:10.1145/3448016.3457550

[74] Jingdong Wang, Ting Zhang, jingkuan song, Nicu Sebe, and Heng Tao Shen. 2018. A Survey on Learning to Hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2018), 769–790. doi:10.1109/TPAMI.2017.2699960

[75] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *Proc. ACM Manag. Data* 2, 1, Article 14 (mar 2024), 27 pages. doi:10.1145/3639269

[76] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 14, 11 (jul 2021), 1964–1978. doi:10.14778/3476249.3476255

[77] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph-and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions. *Data Engineering* (2023), 3–21.

[78] Zeyu Wang, Haoran Xiong, Zhenying He, Peng Wang, and Wei wang. 2024. Distance Comparison Operators for Approximate Nearest Neighbor Search: Exploration and Benchmark. arXiv:2403.13491 [cs.DB] https://arxiv.org/abs/2403.13491

[79] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data*

Bases (VLDB '98). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 194–205.

[80] Mingyu Yang, Wentao Li, Jiabao Jin, Xiaoyao Zhong, Xiangyu Wang, Zhitao Shen, Wei Jia, and Wei Wang. 2024. Effective and General Distance Computation for Approximate Nearest Neighbor Search. arXiv:2404.16322 [cs.DB] https://arxiv.org/abs/2404.16322

[81] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2241–2253. doi:10.1145/3318464.3386131

[82] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. 2010. *Similarity Search: The Metric Space Approach* (1st ed.). Springer Publishing Company, Incorporated.

[83] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite Quantization for Approximate Nearest Neighbor Search. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 32)*, Eric P. Xing and Tony Jebara (Eds.). PMLR, Bejing, China, 838–846. https://proceedings.mlr.press/v32/zhangd14.html

[84] Zilliz. 2023. Pyglass - Graph Library for Approximate Similarity Search. https://github.com/zilliztech/pyglass. Accessed: 17 Apr, 2024.

[85] Zilliz. 2024. Knowhere. https://github.com/zilliztech/knowhere.

[86] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 1, Article 69 (mar 2024), 26 pages. doi:10.1145/3639324

Received October 2024; revised January 2025; accepted February 2025