# RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search

JIANYANG GAO, Nanyang Technological University, Singapore
CHENG LONG*, Nanyang Technological University, Singapore

Searching for approximate nearest neighbors (ANN) in the high-dimensional Euclidean space is a pivotal problem. Recently, with the help of fast SIMD-based implementations, Product Quantization (PQ) and its variants can often efficiently and accurately estimate the distances between the vectors and have achieved great success in the in-memory ANN search. Despite their empirical success, we note that these methods do not have a theoretical error bound and are observed to fail disastrously on some real-world datasets. Motivated by this, we propose a new randomized quantization method named RaBitQ, which quantizes $D$-dimensional vectors into $D$-bit strings. RaBitQ guarantees a sharp theoretical error bound and provides good empirical accuracy at the same time. In addition, we introduce efficient implementations of RaBitQ, supporting to estimate the distances with bitwise operations or SIMD-based operations. Extensive experiments on real-world datasets confirm that (1) our method outperforms PQ and its variants in terms of accuracy-efficiency trade-off by a clear margin and (2) its empirical performance is well-aligned with our theoretical analysis.

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 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 (SIGMOD), Article 167 (June 2024), 27 pages. https://doi.org/10.1145/3654970

## 1 INTRODUCTION

Searching for the nearest neighbor (NN) in the high-dimensional Euclidean space is pivotal for various applications such as information retrieval [60], data mining [16], and recommendations [76]. However, the curse of dimensionality [39, 89] makes exact NN queries on extensive vector databases practically infeasible due to their long response time. To strike a balance between time and accuracy, researchers often explore its relaxed counterpart, known as approximate nearest neighbor (ANN) search [18, 34, 37, 45, 65, 70].

Product Quantization (PQ) and its variants are a family of popular methods for ANN [8, 34, 36, 45, 66–68, 82, 84, 94]. These methods target to efficiently estimate the distances between the data vectors and query vectors during the query phase to shortlist a list of candidates, which would

*Cheng Long is the corresponding author.

Authors' addresses: Jianyang Gao, jianyang.gao@ntu.edu.sg, Nanyang Technological University, Singapore; Cheng Long, c.long@ntu.edu.sg, Nanyang Technological University, Singapore.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike International 4.0 License.

© 2024 Copyright held by the owner/author(s).
ACM 2836-6573/2024/6-ART167
https://doi.org/10.1145/3654970

then be re-ranked based on exact distances for finding the NN. Specifically, during the index phase, they (1) construct a quantization codebook and (2) find for each data vector the nearest vector in the codebook as its quantized vector. The quantized vector is represented and stored as a short quantization code (e.g., the ID of the quantized data vector in the codebook). During the query phase, they (1) pre-compute the (squared) distances [1] between the query and the vectors in the codebook when a query comes and (2) for a data vector, they adopt the distances between the query vector and its quantized data vector (which can be computed by looking up the pre-computed values) as the estimated distances. Recently, with the help of the fast SIMD-based implementation [4, 5], PQ has achieved great success in the in-memory ANN search [37, 43, 48]. In particular, on many real-world datasets, the method can efficiently estimate the distances with high accuracy.

Despite their empirical success on many real-world datasets, to the best of our knowledge, none of PQ and its variants [8, 34, 36, 45, 66–68, 82, 84, 94] provide theoretical error bounds on the estimated distances. This is because they lose guarantees in both (1) the codebook construction component and (2) the distance estimation component. Codebook Construction: They construct the codebook often via approximately solving an optimization problem for a heuristic objective function, e.g., PQ conducts KMeans clustering on the sub-segments of the data vectors and uses the set of the products of cluster centroids as the codebook. However, due to their heuristic nature, it is often difficult to analyze their results theoretically (e.g., no theoretical results have been achieved on the distance between a data vector and its nearest vector in the codebook (i.e., its quantized vector)). Distance Estimation: They estimate the distance between a data vector and a query vector with that between the quantized data vector and the query vector, i.e., they simply treat the quantized data vector as the data vector for computing the distance. While this looks intuitive, it does not come with a theoretical error bound on the approximation. The lack of a theoretical error bound indicates that these methods may unpredictably fail anytime, moderately or severely, when they are deployed in real-world systems to handle new datasets and queries which they have not been tested on. In fact, such failure has been observed on public real-world datasets which are widely adopted to benchmark ANN search. For example, on the dataset MSong, PQ (with the fast SIMD-based implementation [4, 5]) incurs more than 50% of average relative error on the estimated distances between the query and data vectors, which causes disastrous recall of ANN search (e.g., it has no more than 60% recall even with re-ranking applied, as shown in Section 5.2.3).

Table 1. Comparison between RaBitQ and PQ and its variants. More ★'s indicates better efficiency.

| | RaBitQ (new) | PQ and its variants |
|---|---|---|
| Codebook | Randomly transformed bi-valued vectors. | Cartesian product of sub-codebooks. |
| Quantization Code | A bit string. | A sequence of 4-bit/8-bit unsigned integers. |
| Distance Estimator | Unbiased and provides a sharp error bound. | Biased and provide no error bound. |
| Implementation (single) | Bitwise operations. ★★ | Looking up tables in RAM. ★ |
| Implementation (batch) | Fast SIMD-based operations. ★★★ | Fast SIMD-based operations. ★★★ |

In this paper, we propose a new quantization method, which provides unbiased estimation on the distances and achieves a sharp [2] theoretical error bound. The new method achieves this with careful and integrated design in both the codebook construction and distance estimation components. Codebook Construction: It first normalizes the data vectors in order to align them on the unit hypersphere in the $D$-dimensional space. It then constructs the codebook by (1) constructing a set of $2^D$ *bi-valued* vectors whose coordinates are $-1/\sqrt{D}$ or $+1/\sqrt{D}$ (i.e., the set consists of the

[1] By distances, we refer to the squared distances without further specification.

[2] The error bound is sharp in the sense that it achieves the asymptotic optimality shown in [3]. Detailed discussions can be found in Section 3.2.2.

vertices of a hypercube, which evenly spread on the unit hypersphere) and (2) randomly rotating the bi-valued vectors by multiplying each with a *random orthogonal matrix* [3] (i.e., it performs a type of Johnson-Lindenstrauss Transformation [49], JLT in short). For each data vector, its nearest vector from the codebook is taken as the quantized vector. Since each quantized vector is a rotated $D$-dimensional bi-valued vector, we represent its quantization code as a bit string of length $D$, where 0 and 1 indicate the two distinct values. The rationale of the codebook construction is that it has a clear geometric interpretation (i.e., the vectors in the codebook are a set of randomly rotated vectors on the unit hypersphere) such that it is possible to analyze the geometric relationship among the data vectors, their quantized vectors and the query vectors explicitly. Distance Estimation: We carefully design an estimator of the distance between a data vector and a query vector by leveraging the aforementioned geometric relationship. We prove that the estimator is *unbiased* and has a sharp probabilistic error bound with the help of plentiful theoretical tools about the JLT [28, 52, 81]. This is in contrast to PQ and its variants, which simply treat the quantized vector as the data vector for estimating the distances, which is *biased* and provides *no* theoretical error bound. We call the new quantization method, which uses randomly transformed bi-valued vectors for quantizing data vectors, RaBitQ. Compared with PQ and its variants, RaBitQ has its superiority not only in providing error bounds in theory, but also in estimating the distances with smaller empirical errors even with shorter quantization codes by roughly a half (as verified on all the tested datasets shown in Section 5.2.1).

We further introduce two efficient implementations for computing the value of RaBitQ's distance estimator, namely one for a *single* data vector and the other for a *batch* of data vectors. For the former, our implementation is based on simple bitwise operations - recall that our quantization codes are bit strings. Our implementation is on average 3x faster than the original implementation of PQ which relies on looking up tables in RAM while reaching the same accuracy (as shown in Section 5.2.1). Note that for a single data vector, the SIMD-based implementation of PQ [4, 5] is not feasible as it requires to pack the quantization codes in a batch and reorganize their layout carefully. For the latter, the same strategy of the fast SIMD-based implementation [4, 5] can be adopted seamlessly, and thus it achieves similar efficiency as existing SIMD-based implementation of PQ does when similar length quantization codes are used - in this case, our method would provide more accurate estimated distances as explained earlier. Table 1 provides some comparison between RaBitQ and PQ and its variants.

We summarize our major contributions as follows.

(1) We propose a new quantization method, namely RaBitQ. (1) It constructs the codebook via randomly transforming bi-valued vectors. (2) It designs an *unbiased* distance estimator with a sharp probabilistic error bound.
(2) We introduce efficient implementations of computing the distance estimator for RaBitQ. Our implementation is more efficient than its counterpart of PQ and its variants when estimating the distance for a single data vector and is comparably fast when estimating the distances for a batch of data vectors with quantization codes of similar lengths.
(3) We conduct extensive experiments on real-world datasets, which show that (1) RaBitQ provides more accurate estimated distances than PQ (and its variants) even when the former uses shorter codes than the latter by roughly a half (which implies the accuracy gap would be further larger when both methods use codes of similar lengths); (2) RaBitQ works stably well on all datasets tested including some on which PQ (and its variants) fail (which is well aligned with the theoretical results); (3) RaBitQ is superior over PQ (and its variants) in terms

[3]We note that we do not explicitly materialize the codebook, but maintain it conceptually, as existing quantization methods such as PQ do.

of time-accuracy trade-offs for in-memory ANN by a clear margin on all datasets tested; and (4) RaBitQ has its empirical performance well-aligned with the theoretical analysis.

The remainder of the paper is organized as follows. Section 2 introduces the ANN search and PQ and its variants. Section 3 presents our RaBitQ method. Section 4 illustrates the application of 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 ANN QUERY AND QUANTIZATION

**ANN Query.** Suppose that we have a database of $N$ data vectors in the $D$-dimensional Euclidean space. The approximate nearest neighbor (ANN) search query is to retrieve the nearest vector from the database for a given query vector $\mathbf{q}$. The question is usually extended to the query of retrieving the $K$ nearest neighbors. For the ease of narrative, we assume that $K = 1$ in our algorithm description, while all of the proposed techniques can be easily adapted to a general $K$. We focus on the in-memory ANN, which assumes that all the raw data vectors and indexes can be hosted in the main memory [4–6, 30, 32, 58, 65].

**Product Quantization.** Product Quantization (PQ) and its variants are a family of popular methods for ANN [8, 34, 36, 45, 66–68, 82, 84, 94] (for the discussion on a broader range of quantization methods, see Section 6). For a query vector and a data vector, these methods target to efficiently estimate their distance based on some pre-computed short quantization codes. Specifically, for PQ, it splits the $D$-dimensional vectors into $M$ sub-segments (each sub-segment has $D/M$ dimensions). For each sub-segment, it performs KMeans clustering on the $D/M$-dimensional vectors to obtain $2^k$ clusters and then takes the centroids of the clusters as a sub-codebook where $k$ is a tunable parameter which controls the size of the sub-codebook ($k = 8$ by default). The codebook of PQ is then formed by the Cartesian product of the sub-codebooks of the sub-segments and thus has the size of $(2^k)^M$. Correspondingly each quantization code can be represented as an $M$-sized sequence of $k$-bit unsigned integers. During the query phase, asymmetric distance computation is adopted to estimate the distance [45]. In particular, it pre-processes $M$ look-up-tables (LUTs) for each sub-codebook when a query comes. The $i$th LUT contains $2^k$ numbers which represent the squared distances between the vectors in the $i$th sub-codebook and $i$th sub-segment of the query vector. For a given quantization code, by looking up and accumulating the values in the LUTs for $M$ times, PQ can compute an estimated distance.

Recently, [4, 5] propose a SIMD-based fast implementation for PQ (PQ Fast Scan, PQx4fs in short). They speed up the look-up and accumulation operations significantly, making PQ an important component in many popular libraries for in-memory ANN search such as Faiss from Meta [48], ScaNN from Google [37] and NGT-QG from Yahoo Japan [43]. At its core, unlike the original implementation of PQ which relies on looking up the LUTs in RAM, [4, 5] propose to host the LUTs in SIMD registers and look up the LUTs with the SIMD shuffle instructions. To achieve so, the method makes several modifications on PQ. First, in order to fit the LUTs into the AVX2 256-bit registers, it modifies the original setting of $k = 8$ to $k = 4$ so that in each LUT, there are only $2^4$ floating-point numbers. It further quantizes the numbers in the LUT to be 8-bit unsigned integers so that one LUT takes the space of only 128 ($2^4 \times 8$) bits. Thus, one AVX2 256-bit register is able to host two LUTs. Second, in order to look up the LUTs efficiently, the method packs every 32 quantization codes in a batch and reorganizes their layout. In this case, a series of operations can estimate the distances for 32 data vectors all at once. Without further specification, by PQ, we refer to PQx4fs by default because without the fast SIMD-based implementation, the efficiency of PQ is much less competitive in the in-memory ANN search [4, 5] (see Section 5.2.1).

Table 2. Notations.

| Notation | Definition |
|---|---|
| $\mathbf{o}_r, \mathbf{q}_r$ | The raw data and query vectors. |
| $\mathbf{o}, \mathbf{q}$ | The normalized data and query vectors. |
| $C, C_{rand}$ | The quantization codebook, its randomized version. |
| $P$ | A random orthogonal transformation matrix. |
| $\bar{\mathbf{x}}$ | The code in $C$ s.t. $P\bar{\mathbf{x}}$ is the quantized vector of $\mathbf{o}$. |
| $\bar{\mathbf{o}}$ | The quantized vector of $\mathbf{o}$ in $C_{rand}$, i.e., $\bar{\mathbf{o}} = P\bar{\mathbf{x}}$. |
| $\bar{\mathbf{x}}_b$ | The quantization code of $\mathbf{o}$ as a $D$-bit string. |
| $\mathbf{q}'$ | The inversely transformed query vector, i.e., $P^{-1}\mathbf{q}$. |
| $\bar{\mathbf{q}}$ | The quantized query vector of $\mathbf{q}'$. |
| $\bar{\mathbf{q}}_u$ | The unsigned integer representation of $\bar{\mathbf{q}}$. |

Nevertheless, none of PQ and its variants provide a theoretical error bound on the errors of the estimated distances [8, 34, 36, 45, 66–68, 82, 84, 94], as explained in Section 1. Indeed, we find that the accuracy of PQ can be disastrous (see Section 5.2.3), e.g., on the dataset MSong, PQ cannot achieve $\geq 60\%$ recall even with re-ranking applied. We note that Locality Sensitive Hashing (LSH) is a family of methods which promise rigorous theoretical guarantee [18, 38, 39, 77–79]. However, as is widely reported [6, 58, 84], these methods can hardly produce competitive empirical performance. Furthermore, their guarantees are on the accuracy of $c$-approximate NN query. In particular, LSH guarantees to return a data vector whose distance from the query is at most $(1 + c)$ times of a fixed radius $r$ with high probability (if there exists a data vector whose distance from the query is within the radius $r$). Due to the relaxation factor $c$, there can be many that satisfy the statement. The guarantee of returning *any* of them does not help to produce high recall for ANN search. In contrast, a guarantee on the distance estimation can help to decide whether a data vector should be re-ranked for achieving high recall (see Section 4).

## 3 THE RABITQ METHOD

In this section, we present the details of RaBitQ. In Section 3.1, we present the index phase of RaBitQ, which normalizes the data vectors (Section 3.1.1), constructs a codebook (Section 3.1.2) and computes the quantized vectors of data vectors (Section 3.1.3). In Section 3.2, we introduce the distance estimator of RaBitQ, which is unbiased and provides a rigorous theoretical error bound. In Section 3.3, we illustrate how to efficiently compute the value of the estimator. In Section 3.4, we summarize the RaBitQ method. Table 2 lists the frequently used notations and their definitions.

### 3.1 Quantizing the Data Vectors with RaBitQ

*3.1.1 Converting the Raw Vectors into Unit Vectors via Normalization.* We note that directly constructing a codebook for the raw data vectors is challenging for achieving the theoretical error bound because the Euclidean space is *unbounded* and the raw data vectors may appear anywhere in the infinitely large space. To deal with this issue, a natural idea is to normalize the raw vectors into *unit* vectors. Specifically, let $\mathbf{c}$ be the centroid of the raw data vectors. We normalize the raw data vectors $\mathbf{o}_r$ to be $\mathbf{o} := \frac{\mathbf{o}_r - \mathbf{c}}{\|\mathbf{o}_r - \mathbf{c}\|}$. Similarly, we normalize the raw query vector $\mathbf{q}_r$ (when it comes in the query phase) to be $\mathbf{q} := \frac{\mathbf{q}_r - \mathbf{c}}{\|\mathbf{q}_r - \mathbf{c}\|}$. The following expressions bridge the distance between the raw vectors (i.e., our target) and the inner product of the normalized vectors.

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

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

We note that $\|\mathbf{o}_r - \mathbf{c}\|$ is the distance from the data vector to the centroid, which can be pre-computed during the index phase. $\|\mathbf{q}_r - \mathbf{c}\|$ is the distance from the query vector to the centroid. It can be computed during the query phase and its cost can be shared by all the data vectors. Thus, based on Equation (2), the question of computing $\|\mathbf{o}_r - \mathbf{q}_r\|^2$ is reduced to that of computing the inner product of two unit vectors $\langle \mathbf{q}, \mathbf{o} \rangle$. We note that in practice we can cluster the data vectors first (e.g., via KMeans clustering) and perform the normalization for data vectors within a cluster individually based on the centroid of the cluster. When considering the data vectors within a cluster, we normalize the query vector based on the corresponding centroid. In this way, the normalized data vectors are expected to spread evenly on the unit hypersphere, removing the skewness of the data (if any) to some extent. For the sake of convenience, in the following parts without further clarification, by the data and query vector, we refer to their corresponding unit vectors. With this conversion, we next focus on estimating the inner product of the unit vectors, i.e., $\langle \mathbf{q}, \mathbf{o} \rangle$.

*3.1.2 Constructing the Codebook.* As mentioned in Section 3.1.1, the data vectors are supposed, to some extent, to be evenly spreading on the unit hypersphere due to the normalization. By intuition, our codebook should also spread evenly on the unit hypersphere. To this end, a natural construction of the codebook is given as follows.

$$C := \left\{ +\frac{1}{\sqrt{D}}, -\frac{1}{\sqrt{D}} \right\}^D \tag{3}$$

It is easy to verify that the vectors in $C$ are unit vectors and the codebook has the size of $|C| = 2^D$.

However, such construction may favor some certain vectors and perform poorly for others. For example, for the data vector $(1/\sqrt{D}, ..., 1/\sqrt{D})$, its quantized data vector (which corresponds to the vector in $C$ closest from the data vector) is $(1/\sqrt{D}, ..., 1/\sqrt{D})$, and its squared distance to the quantized data vector is 0. In contrast, for the vector $(1, 0, ..., 0)$, its quantized data vector is also $(1/\sqrt{D}, ..., 1/\sqrt{D})$, and its squared distance to the quantized data vector equals to $2 - 2/\sqrt{D}$. To deal with this issue, we inject the codebook some randomness. Specifically, let $P$ be a random orthogonal matrix. We propose to apply the transformation $P$ to the codebook (which is one type of the Johnson-Lindenstrauss Transformation [49]). Our final codebook is given as follows.

$$C_{rand} := \{P\mathbf{x} | \mathbf{x} \in C\} \tag{4}$$

Geometrically, the transformation simply rotates the codebook because the matrix $P$ is orthogonal, and thus, the vectors in $C_{rand}$ are still unit vectors. Moreover, the rotation is uniformly sampled from "all the possible rotations" of the space. Thus, for a unit vector in the codebook $C$, it has equal probability to be rotated to anywhere on the unit hypersphere. This step thus removes the preference of the deterministic codebook $C$ on specific vectors.

We note that to construct the codebook $C_{rand}$, we only need to sample a random transformation matrix $P$. To store the codebook $C_{rand}$, we only need to physically store the sampled $P$ but not all the transformed vectors. The codebook constructed by this operation is much simpler than its counterpart in PQ and its variants which rely on approximately solving an optimization problem.

*3.1.3 Computing the Quantized Codes of Data Vectors.* With the constructed codebook, the next step is to find the nearest vector from $C_{rand}$ for each data vector as its quantized vector. For a *unit* vector $\mathbf{o}$, to find its nearest vector, it is equivalent to find the one which has the largest inner product with it. Let $P\bar{\mathbf{x}} \in C_{rand}$ be the quantized data vector (where $\bar{\mathbf{x}} \in C$). The following equations

illustrate the idea rigorously.

$$\bar{\mathbf{x}} = \arg\min_{\mathbf{x}\in C} \|\mathbf{o} - P\mathbf{x}\|^2 \tag{5}$$

$$= \arg\min_{\mathbf{x}\in C} (\|\mathbf{o}\|^2 + \|P\mathbf{x}\|^2 - 2\langle \mathbf{o}, P\mathbf{x}\rangle) \tag{6}$$

$$= \arg\min_{\mathbf{x}\in C} (2 - 2\langle \mathbf{o}, P\mathbf{x}\rangle) = \arg\max_{\mathbf{x}\in C} \langle \mathbf{o}, P\mathbf{x}\rangle \tag{7}$$

Equation (5) is based on the definition of the quantized data vector. Equation (6) is due to elementary linear algebra operations. Equation (7) is because $P\mathbf{x}$ and $\mathbf{o}$ are unit vectors. However, by Equation (7), it is costly to find the quantized data vector by physically transforming the huge codebook and finding the nearest vector via enumeration. We note that the inner product is invariant to orthogonal transformation (i.e., rotation). Thus, instead of transforming the huge codebook, we *inversely* transform the data vector $\mathbf{o}$. The following expressions formally present the idea.

$$\langle \mathbf{o}, P\mathbf{x}\rangle = \left\langle P^{-1}\mathbf{o}, P^{-1}P\mathbf{x}\right\rangle = \left\langle P^{-1}\mathbf{o}, \mathbf{x}\right\rangle \tag{8}$$

Recall that the entries of $\mathbf{x} \in C$ are $\pm 1/\sqrt{D}$. To maximize the inner product, we only need to pick the $\bar{\mathbf{x}} \in C$ whose signs of the entries match those of $P^{-1}\mathbf{o}$. Then $P\bar{\mathbf{x}}$ is the quantized data vector.

In summary, to find the nearest vector of a data vector $\mathbf{o}$ from $C_{rand}$, we can inversely transform $\mathbf{o}$ with $P^{-1}$ and store the signs of its entries as a $D$-bit string $\bar{\mathbf{x}}_b \in \{0,1\}^D$. We call the stored binary string $\bar{\mathbf{x}}_b$ as the *quantization code*, which can be used to re-construct the quantized vector $\bar{\mathbf{x}}$. Let $\mathbf{1}_D$ be the $D$-dimensional vector which has all its entries being ones. The relationship between $\bar{\mathbf{x}}_b$ and $\bar{\mathbf{x}}$ is given as $\bar{\mathbf{x}} = (2\bar{\mathbf{x}}_b - \mathbf{1}_D)/\sqrt{D}$, i.e., when the $i$th coordinate $\bar{\mathbf{x}}_b[i] = 1$, we have $\bar{\mathbf{x}}[i] = 1/\sqrt{D}$ and when $\bar{\mathbf{x}}_b[i] = 0$, we have $\bar{\mathbf{x}}[i] = -1/\sqrt{D}$. For the sake of convenience, we denote the quantized data vector as $\bar{\mathbf{o}} := P\bar{\mathbf{x}}$.

Till now, we have finished the pre-processing in the index phase. We note that the time cost in the index phase is not a bottleneck for our method, which is the same as in the cases of PQ and OPQ (a popular variant of PQ) [34]. For example, on the dataset GIST with one million 960-dimensional vectors, with 32 threads on CPU, our method, PQ and OPQ take 117s, 105s and 291s respectively. The space complexity of the methods is not a bottleneck for the in-memory ANN either, because the space consumption is largely due to the space for storing the raw vectors. As a comparison, each raw vector takes $32D$ bits (i.e., $D$ floating-point numbers). Our method by default has $D$ bits for a quantization code. PQ and OPQ by default have $2D$ bits for a quantization code (i.e., $M = D/2$) according to [25, 37], which is significantly smaller than the space for storing the raw vectors.

### 3.2 Constructing an Unbiased Estimator

Recall that the problem of computing $\|\mathbf{o}_r - \mathbf{q}_r\|^2$ can be reduced to that of computing the inner product of two unit vectors $\langle \mathbf{o}, \mathbf{q}\rangle$. In this section, we introduce an unbiased estimator for $\langle \mathbf{o}, \mathbf{q}\rangle$. Unlike PQ and its variants which simply treat the quantized data vector as the data vector for estimating the distances without theoretical error bounds, we first explicitly derive the relationship between $\langle \mathbf{o}, \mathbf{q}\rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{q}\rangle$ in Section 3.2.1. We then construct an unbiased estimator for $\langle \mathbf{o}, \mathbf{q}\rangle$ based on the derived relationships and present its rigorous error bound in Section 3.2.2.

*3.2.1 Analyzing the Explicit Relationship between $\langle \mathbf{o}, \mathbf{q}\rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{q}\rangle$.* We note that the relationship between $\langle \mathbf{o}, \mathbf{q}\rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{q}\rangle$ depends only on the projection of $\bar{\mathbf{o}}$ on the two-dimensional subspace spanned by $\mathbf{o}$ and $\mathbf{q}$, which is illustrated on the left panel of Figure 1. For the component of $\bar{\mathbf{o}}$ which is perpendicular to the subspace, it has no effect on the inner product $\langle \bar{\mathbf{o}}, \mathbf{q}\rangle$. The following lemma presents the specific result. The proof can be found in the technical report [33].

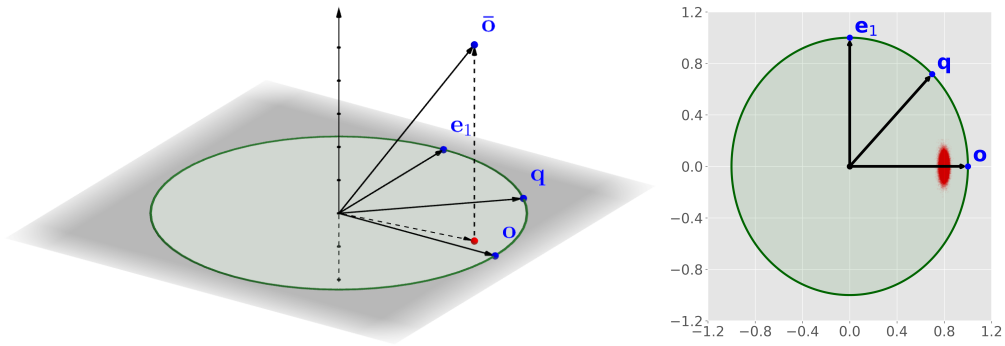


Fig. 1. Geometric Relationship among the Vectors.

Lemma 3.1 (Geometric Relationship). *Let* $\mathbf{o}, \mathbf{q}$ *and* $\bar{\mathbf{o}}$ *be any three unit vectors. When* $\mathbf{o}$ *and* $\mathbf{q}$ *are collinear (i.e.,* $\mathbf{o} = \mathbf{q}$ *or* $\mathbf{o} = -\mathbf{q}$*), we have*

$$\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{o}, \mathbf{q} \rangle \tag{9}$$

*When* $\mathbf{o}$ *and* $\mathbf{q}$ *are non-collinear, we have*

$$\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{o}, \mathbf{q} \rangle + \langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle \cdot \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2} \tag{10}$$

*where* $\mathbf{e}_1$ *is* $\mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o}$ *with its norm normalized to be 1, i.e.,* $\mathbf{e}_1 := \frac{\mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o}}{\|\mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o}\|}$*. We note that* $\mathbf{o} \perp \mathbf{e}_1$ *(since* $\langle \mathbf{o}, \mathbf{e}_1 \rangle = 0$*) and* $\|\mathbf{e}_1\| = 1$*.*

Recall that we target to estimate $\langle \mathbf{o}, \mathbf{q} \rangle$. If we exactly know the values of all the variables other than $\langle \mathbf{o}, \mathbf{q} \rangle$, we can compute the exact value of $\langle \mathbf{o}, \mathbf{q} \rangle$ by solving Equations (9) and (10). In particular, in Equations (9) and (10), $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ is the inner product between the quantized data vector and the data vector. Its value can be pre-computed in the index phase. $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ is the inner product between the quantized data vector and the query vector. Its value can be efficiently computed in the query phase (we will specify in Section 3.3 how it can be efficiently computed). Thus, when $\mathbf{o}$ and $\mathbf{q}$ are collinear, we can compute the value of $\langle \mathbf{o}, \mathbf{q} \rangle$ exactly by solving Equation (9), i.e., $\langle \mathbf{o}, \mathbf{q} \rangle = \frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$.

When $\mathbf{o}$ and $\mathbf{q}$ are non-collinear (which is a more common case), in order to exactly solve the Equation (10), we need to know the value of $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$. However, as $\mathbf{e}_1$ depends on both $\mathbf{o}$ and $\mathbf{q}$ (which can be seen by its definition), $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ can be neither pre-computed in the index phase (because it depends on $\mathbf{q}$) nor computed efficiently in the query phase without accessing $\mathbf{o}$.

We notice that although we cannot efficiently compute the exact value of $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ [4], given the random nature of $\bar{\mathbf{o}}$, we explicitly know its distribution. Specifically, recall that we have sampled a random orthogonal matrix $P$, applied it to the codebook $C$ and generated a *randomized* codebook $C_{rand}$. $\bar{\mathbf{o}}$ is a vector picked from the randomized codebook $C_{rand}$ and thus, it is a random vector. $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ correspond to the projection of the random vector $\bar{\mathbf{o}}$ onto two fixed directions (i.e., the directions are $\mathbf{o}$ and $\mathbf{e}_1$, where $\mathbf{o} \perp \mathbf{e}_1$). Thus, they are mutually correlated random variables.

We rigorously analyze the distributions of the random variables. The core conclusions of the analysis are briefly summarized as follows while the detailed presentation and proof are left in the technical report [33] due to the page limit. Specifically, our analysis indicates that when $D$

[4]In particular, when we say "computing the value" of a random variable, it refers to computing its *observed* value based on a certain sampled $P$.

ranges from $10^2$ to $10^6$, it is always true that $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ has the expectation [5] of around 0.8 and $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ has the expectation of exactly 0. It further indicates that, with high probability, these random variables would not deviate from their expectation by $\Omega(1/\sqrt{D})$. This conclusion quantitatively presents the extent that the random variables concentrate around their expected values, which will be used later for analyzing the error bound of our estimator. To empirically verify our analysis, we repeatedly and independently sample the random orthogonal matrices $P$ $10^5$ times for a pair of fixed $\mathbf{o}, \mathbf{q}$ in the 128-dimensional space. The right panel of Figure 1 visualizes the projection of $\bar{\mathbf{o}}$ on the 2-dimensional space spanned by $\mathbf{o}, \mathbf{q}$ with the red point cloud (each point represents the projection of an $\bar{\mathbf{o}}$ based on a sampled random matrix $P$). In particular, $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ (the x-axis) is shown to be concentrated around 0.8. $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ (the y-axis) is concentrated and symmetrically distributed around 0, which verifies our theoretical analysis perfectly.

*3.2.2 Constructing an Unbiased Estimator for $\langle \mathbf{o}, \mathbf{q} \rangle$.* Based on our analysis on Equation (9), for the case that $\mathbf{o}, \mathbf{q}$ are collinear, $\langle \mathbf{o}, \mathbf{q} \rangle$ can be explicitly solved by $\langle \mathbf{o}, \mathbf{q} \rangle = \frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$. Thus, it is natural to conjecture that for the case that $\mathbf{o}, \mathbf{q}$ are non-collinear, $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ should also be a good estimator for $\langle \mathbf{o}, \mathbf{q} \rangle$. We thus deduce from it as follows.

$$\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} = \frac{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{o}, \mathbf{q} \rangle + \langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle \cdot \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2}}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} \tag{11}$$

$$= \langle \mathbf{o}, \mathbf{q} \rangle + \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2} \cdot \frac{\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} \tag{12}$$

where Equation (11) is by Equation (10) and Equation (12) simplifies Equation (11). We note that the last term in Equation (12) can be viewed as the error term of the estimator. Recall that based on our analysis in Section 3.2.1, $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ is concentrated around 0.8. $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ has the expectation of 0 and is concentrated. It implies that the error term has 0 expectation and will not deviate largely from 0 due to the concentration. The following theorem presents the specific results. The rigorous proof can be found in the technical report [33].

Theorem 3.2 (Estimator). *The unbiasedness is given as*

$$\mathbb{E}\left[\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}\right] = \langle \mathbf{o}, \mathbf{q} \rangle \tag{13}$$

*The error bound of the estimator is given as*

$$\mathbb{P}\left\{\left|\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} - \langle \mathbf{o}, \mathbf{q} \rangle\right| > \sqrt{\frac{1 - \langle \bar{\mathbf{o}}, \mathbf{o} \rangle^2}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle^2}} \cdot \frac{\epsilon_0}{\sqrt{D-1}}\right\} \leq 2e^{-c_0 \epsilon_0^2} \tag{14}$$

*where $\epsilon_0$ is a parameter which controls the failure probability. $c_0$ is a constant factor. The error bound can be concisely presented as*

$$\left|\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} - \langle \mathbf{o}, \mathbf{q} \rangle\right| = O\left(\frac{1}{\sqrt{D}}\right) \text{ with high probability} \tag{15}$$

Due to Equations (2) and (13), the unbiased estimator of $\langle \mathbf{o}, \mathbf{q} \rangle$ can further induce an unbiased estimator of the squared distance between the raw data and query vectors. We provide empirical verification on the unbiasedness in Section 5.2.6. Besides, we would like to highlight that based on

[5] The exact expected value is $\mathbb{E}\left[\langle \bar{\mathbf{o}}, \mathbf{o} \rangle\right] = \sqrt{\frac{D}{\pi}} \frac{2\Gamma(\frac{D}{2})}{(D-1)\Gamma(\frac{D-1}{2})}$, where $\Gamma(\cdot)$ is the Gamma function. The expected value ranges from 0.798 to 0.800 for $D \in [10^2, 10^6]$.

similar analysis, an alternative estimator $\langle \mathbf{o}, \mathbf{q} \rangle \approx \langle \bar{\mathbf{o}}, \mathbf{q} \rangle$, i.e., by simply treating the quantized data vector as the data vector as PQ does, can be easily proved to be *biased*.

Equation (14) presents the error bound of our estimator. In particular, it presents a $1-2\exp(-c_0\epsilon_0^2)$ confidence interval

$$\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} \pm \sqrt{\frac{1 - \langle \bar{\mathbf{o}}, \mathbf{o} \rangle^2}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle^2}} \cdot \frac{\epsilon_0}{\sqrt{D-1}} \tag{16}$$

We note that the failure probability (i.e., the probability that the confidence interval does not cover the true value of $\langle \mathbf{o}, \mathbf{q} \rangle$) is $2\exp(-c_0\epsilon_0^2)$. It decays in a quadratic-exponential trend wrt $\epsilon_0$, which is extremely fast. The length of the confidence interval grows linearly wrt $\epsilon_0$. Thus, $\epsilon_0 = \Theta(\sqrt{\log(1/\delta)})$ corresponds to a failure probability of at most $\delta$, which indicates that a short confidence interval can correspond to a high confidence level. In practice, $\epsilon_0$ is fixed to be 1.9 in pursuit of nearly perfect confidence (see Section 5.2.4 for the empirical verification study). Recall that $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ is concentrated around 0.8. Based on the values of $\epsilon_0$ and $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$, the error bound can be further concisely presented as Equation (15), i.e., it guarantees an error bound of $O(1/\sqrt{D})$. According to a recent theoretical study [3], for $D$-dimensional vectors, with a short code of $D$ bits, it is *impossible* in theory for a method to provide a bound which is tighter than $O(1/\sqrt{D})$ (the failure probability is viewed as a constant). Thus, Equation (15) indicates that RaBitQ's error bound is sharp, i.e., it achieves the asymptotic optimality. The error bound will be later used in ANN search to determine whether a data vector should be re-ranked (see Section 4).

Furthermore, we note that RaBitQ provides an error bound in an additive form [3] (i.e., absolute error). When the data vectors are well normalized (recall that in Section 3.1.1 we normalize the data vectors), the bound can be pushed forward to a multiplicative form [41] (i.e., relative error). We leave the detailed discussion in the technical report [33] because it is based on an assumption that the data vectors are well normalized. Note that all other theoretical results introduced in this paper do not rely on any assumptions on the data, i.e., the additive bound holds regardless of the data distribution. In the present work, we adopt a simple and natural method of normalization (i.e., with the centroids of IVF as will be introduced in Section 4) to instantiate our scheme of quantization, while we have yet to extensively explore the normalization step itself. We shall leave it as future work to rigorously study the normalization problem.

### 3.3 Computing the Estimator Efficiently

Recall that $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ is the estimator. Since $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ has been pre-computed during the index phase, the remaining question is to compute the value of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ efficiently. For the sake of convenience, we denote $P^{-1}\mathbf{q}$ as $\mathbf{q}'$. Like what we do in Section 3.1.3, in order to compute $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$, we can compute $\langle \bar{\mathbf{x}}, \mathbf{q}' \rangle$, which can be verified as follows.

$$\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \langle P\bar{\mathbf{x}}, \mathbf{q} \rangle = \left\langle P^{-1}P\bar{\mathbf{x}}, P^{-1}\mathbf{q} \right\rangle = \langle \bar{\mathbf{x}}, \mathbf{q}' \rangle \tag{17}$$

*3.3.1 Quantizing the Transformed Query Vector.* Recall that $\bar{\mathbf{x}}$ is a bi-valued vector whose entries are $\pm 1/\sqrt{D}$. It is represented and stored as a binary quantization code $\bar{\mathbf{x}}_b$ as is discussed in Section 3.1.3. $\mathbf{q}'$ is a real-valued vector, whose entries are conventionally represented by floating-point numbers (floats in short). We note that in our method, representing the entries of $\mathbf{q}'$ with floats is an overkill. Specifically, recall that our method adopts $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ as an estimator of $\langle \mathbf{o}, \mathbf{q} \rangle$. Even if we obtain a perfectly accurate result in the computation of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$, our estimation of $\langle \mathbf{o}, \mathbf{q} \rangle$ is still approximate. Thus, instead of exactly computing $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$, we aim to guarantee that the error produced in the computation of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ is much smaller than the error of the estimator itself.

Specifically, we apply uniform scalar quantization on the entries of $\mathbf{q}'$ and represent them as $B_q$-bit unsigned integers. We denote the $i$th entry of the vector $\mathbf{q}'$ as $\mathbf{q}'[i]$. Let $v_l := \min_{1\le i\le D} \mathbf{q}'[i], v_r := \max_{1\le i\le D} \mathbf{q}'[i]$ and $\Delta := (v_r - v_l)/(2^{B_q} - 1)$. The uniform scalar quantization uniformly splits the range of the values $[v_l, v_r]$ into $2^{B_q} - 1$ segments, where each segment has the length of $\Delta$. Then for a value $v = v_l + m \cdot \Delta + t, m = 0, 1, ..., 2^{B_q} - 1, t \in [0, \Delta)$, the method quantizes it by rounding it up to its nearest boundary of the segments (i.e., $v_l + m \cdot \Delta$ or $v_l + (m + 1) \cdot \Delta$) and representing it with the corresponding $B_q$-bit unsigned integer (i.e., $m$ or $m + 1$). Let $\bar{\mathbf{q}}$ be the vector whose entries are equal to the quantized values of the entries of $\mathbf{q}'$ (we term it as the quantized query vector) and $\bar{\mathbf{q}}_u$ be its $B_q$-bit unsigned integer representation, where $\bar{\mathbf{q}} = \Delta \cdot \bar{\mathbf{q}}_u + v_l \cdot \mathbf{1}_D$ (recall that $\mathbf{1}_D$ is the $D$-dimensional vector with all its entries as ones). Then, we can compute $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$ as an approximation of $\langle \bar{\mathbf{x}}, \mathbf{q}' \rangle$.

Furthermore, to retain the theoretical guarantee, we adopt the trick of randomizing the uniform scalar quantization [3, 92]. Specifically, unlike the conventional method which rounds up a value to its nearest boundary of the segments, the randomized method rounds it to its left or right boundary randomly. The rationale is that for a value $v = v_l + m \cdot \Delta + t, m = 0, 1, ..., 2^{B_q} - 1, t \in [0, \Delta)$, when it is rounded to $v_l + m \cdot \Delta$, it will cause an error of under-estimation $-t < 0$. When it is rounded to $v_l + (m + 1) \cdot \Delta$, it will cause an error of over-estimation $\Delta - t > 0$. If we assign $1 - t/\Delta$ probability to the former event and $t/\Delta$ probability to the latter event, the expected error would be 0, which makes the computation unbiased. We note that this operation can be easily achieved by letting

$$\bar{\mathbf{q}}_u[i] := \left\lfloor \frac{\mathbf{q}'[i] - v_l}{\Delta} + u_i \right\rfloor \tag{18}$$

where $u_i$ is sampled from the uniform distribution on $[0, 1]$. Moreover, based on the randomized method, we can analyze the minimum $B_q$ needed for making the error introduced by the uniform scalar quantization negligible. The result is presented with the following theorem. The detailed proof can be found in the technical report [33].

Theorem 3.3. *$B_q = \Theta(\log \log D)$ suffices to guarantee that $|\langle \bar{\mathbf{x}}, \mathbf{q}' \rangle - \langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle| = O(1/\sqrt{D})$ with high probability.*

Recall that the estimator has the error of $O(1/\sqrt{D})$ (see Section 3.2.2). The above theorem shows that setting $B_q = \Theta(\log \log D)$ suffices to guarantee that the error introduced by the uniform scalar quantization is at the same order as the error introduced by estimator. Because the error decreases exponentially wrt $B_q$, increasing $B_q$ by a small constant (i.e., $B_q$ is still at the order of $\Theta(\log \log D)$) guarantees that the error is much smaller than that of the estimator. We provide the empirical verification study for $B_q$ in Section 5.2.5. The result shows that when $B_q = 4$, the error introduced by the uniform scalar quantization would be negligible.

*3.3.2 Computing $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$ Efficiently.* We next present how to compute $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$ efficiently. We first express $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$ with $\bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u$ as follows.

$$\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle = \left\langle \frac{2\bar{\mathbf{x}}_b - \mathbf{1}_D}{\sqrt{D}}, \Delta \cdot \bar{\mathbf{q}}_u + v_l \cdot \mathbf{1}_D \right\rangle \tag{19}$$

$$= \frac{2\Delta}{\sqrt{D}} \langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle + \frac{2v_l}{\sqrt{D}} \sum_{i=1}^{D} \bar{\mathbf{x}}_b[i] - \frac{\Delta}{\sqrt{D}} \sum_{i=1}^{D} \bar{\mathbf{q}}_u[i] - \sqrt{D} \cdot v_l \tag{20}$$

Note that the factors $\Delta$ and $v_l$ are known when we quantize the query vector. $\sum_{i=1}^{D} \bar{\mathbf{x}}_b[i]$ corresponds to the number of 1's in the bit string $\bar{\mathbf{x}}_b$, which can be pre-computed during the index phase. $\sum_{i=1}^{D} \bar{\mathbf{q}}_u[i]$ depends only on the query vector. Its cost of computation can be shared by all the data

vectors. The remaining task is to compute $\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$ where the coordinates of $\bar{\mathbf{x}}_b$ are 0 or 1 and those of $\bar{\mathbf{q}}_u$ are unsigned $B_q$-bit integers.

We provide two versions of fast computation for $\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$. The first version targets the case of a *single* quantization code, as the original implementation of PQ [45] does. The second version targets the case of a packed *batch* of quantization codes, as the fast SIMD-based implementation of PQ [4, 5] does. We note that in general, both our method and PQ have higher throughput in the second case than that in the first case, i.e., they estimate the distances for more quantization codes within certain time. We note that the second case requires the quantization codes to be packed in a batch, which is feasible in some certain scenarios only.

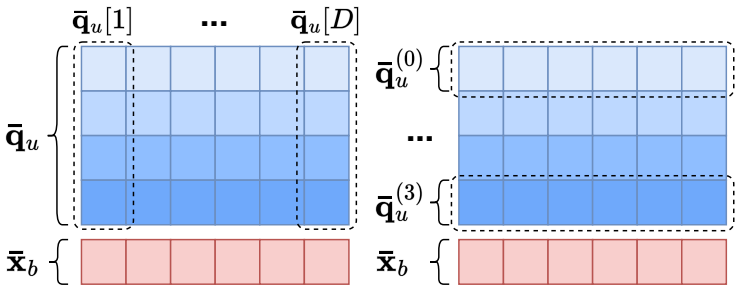


Fig. 2. Bitwise Decomposition of $\bar{\mathbf{q}}_u$.

For the first case where the estimation of the distance is for a query vector and a *single* quantization code, we note that an unsigned $B_q$-bit integer can be decomposed into $B_q$ binary values as shown in Figure 2. The left panel represents the naive computation of $\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$. The right panel represents the proposed bitwise computation of $\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$. Let $\bar{\mathbf{q}}_u^{(j)}[i] \in \{0, 1\}$ be the $j$th bit of $\bar{\mathbf{q}}_u[i]$ where $0 \leq j < B_q$. The following expression specifies the idea.

$$\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle = \sum_{i=1}^{D} \bar{\mathbf{x}}_b[i] \cdot \bar{\mathbf{q}}_u[i] = \sum_{i=1}^{D} \bar{\mathbf{x}}_b[i] \cdot \sum_{j=0}^{B_q-1} \bar{\mathbf{q}}_u^{(j)}[i] \cdot 2^j \tag{21}$$

$$= \sum_{j=0}^{B_q-1} 2^j \cdot \sum_{i=1}^{D} \bar{\mathbf{x}}_b[i] \cdot \bar{\mathbf{q}}_u^{(j)}[i] = \sum_{j=0}^{B_q-1} 2^j \cdot \left\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u^{(j)} \right\rangle \tag{22}$$

Equation (22) shows that $\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$ can be expressed as a weighted sum of the inner product of the binary vectors, i.e., $\left\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u^{(j)} \right\rangle$ for $0 \leq j < B_q$. In particular, we note that the inner product of binary vectors can be efficiently achieved by bitwise operations, i.e., bitwise-and and popcount (a.k.a., bitcount). Thus, the computation of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ is finally reduced to $B_q$ bitwise-and and popcount operations on $D$-bit strings, which are well supported by virtually all platforms. As a comparison, we note that, as is comprehensively studied in [4], PQ relies on looking up LUTs in RAM, which cannot be implemented efficiently. Based on our experiments in Section 5.2.1, on average our method runs 3x faster than PQ and OPQ (a popular variant of PQ [34]) while reaching the same accuracy.

For the second case where the estimation of the distance is for a query vector and a packed *batch* of quantization codes, we note that our method can seamlessly adopt the same fast SIMD-based implementation [4, 5] as PQ does. In particular, for a $D$-bit string, we split it into $D/4$ sub-segments where each sub-segment has 4 bits. We then pre-process $D/4$ LUTs where each LUT has $2^4$ unsigned integers corresponding to the inner products between a sub-segment of $\bar{\mathbf{q}}_u$ and the $2^4$ possible binary strings of a 4-bit sub-segment. For a quantization code of a data vector, we can compute

$\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$ by looking up and accumulating the values in the LUTs for $D/4$ times. We note that the computation is reduced to exactly the form of PQ and thus can adopt the fast SIMD-based implementation seamlessly. Recall that our method has the quantization codes of $D$ bits by default while PQ and OPQ have the codes of $2D$ bits by default. Therefore, our method has better efficiency than PQ and OPQ for computing approximate distances based on quantized vectors. Furthermore, as is shown in Section 5.2.1, in the default setting, our method also achieves consistently better accuracy than PQ and OPQ despite that our method uses a shorter quantization code (i.e., $D$ v.s. $2D$).

### 3.4 Summary of RaBitQ

We summarize the RaBitQ algorithm in Algorithm 1 (the index phase) and Algorithm 2 (the query phase). In the index phase, it takes a set of raw data vectors as inputs. It normalizes the set of vectors based on Section 3.1.1 (line 1), constructs the RaBitQ codebook by sampling a random orthogonal matrix $P$ based on Section 3.1.2 (line 2) and computes the quantization codes $\bar{\mathbf{x}}_b$ based on Section 3.1.3 (line 3). In the query phase, it takes a raw query vector, a set of IDs of the data vectors and the pre-processed variables about the RaBitQ method as inputs. It first inversely transforms, normalizes and quantizes the raw query vector (line 1-2). We note that the time cost of these steps can be shared by all the data vectors. Then for each input ID of the data vectors, it efficiently computes the value of $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ based on Section 3.3.2, adopts it as an unbiased estimation of $\langle \mathbf{o}, \mathbf{q} \rangle$ based on Section 3.2 and further computes an estimated distance between the raw query and the raw data vectors based on Section 3.1.1 (line 3-5).

**Algorithm 1:** RaBitQ (Index Phase)

**Input** : A set of raw data vectors

**Output**: The sampled matrix $P$; the quantization code $\bar{\mathbf{x}}_b$; the pre-computed results of $\|\mathbf{o}_r - \mathbf{c}\|$ and $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$

1 Normalize the set of vectors (Section 3.1.1)
2 Sample a random orthogonal matrix $P$ to construct the codebook $C_{rand}$ (Section 3.1.2)
3 Compute the quantization codes $\bar{\mathbf{x}}_b$ (Section 3.1.3)
4 Pre-compute the values of $\|\mathbf{o}_r - \mathbf{c}\|$ and $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$

**Algorithm 2:** RaBitQ (Query Phase)

**Input** : A raw query vector $\mathbf{q}_r$; the sampled matrix $P$; a set of IDs of the data vectors, their quantization codes $\bar{\mathbf{x}}_b$ and the results of $\|\mathbf{o}_r - \mathbf{c}\|$ and $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$

**Output**: A set of approximate distances between the raw query and the raw data vectors

1 Normalize and inversely transform the raw query vector and obtain $\mathbf{q}'$
2 Quantize $\mathbf{q}'$ into $\bar{\mathbf{q}}$ (Section 3.3.1)
3 **foreach** *input ID of the data vectors* **do**
4  Compute the value of the estimator $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ as an approximation of $\langle \mathbf{o}, \mathbf{q} \rangle$ (Section 3.3.2)
5  Compute an estimated distance between the raw query and the data vector based on Equation (2)

## 4 RABITQ FOR IN-MEMORY ANN SEARCH

Next we present the application of our method to the in-memory ANN search. We note that the popular quantization method PQx4fs has been used in combination with the inverted-file-based indexes such as IVF [45] or the graph-based indexes such as NGT-QG [43] for in-memory ANN search. The combination of a quantization method with IVF can be easily done without much efforts. For example, we can use the quantization method to estimate the distances between the data vectors in the clusters that are probed, which decide those vectors to be re-ranked based on exact distances. In this case, batches of data vectors can be formed and the SIMD-based fast implementation (i.e., PQx4fs) can be adopted. Nevertheless, the combination of a quantization method with graph-based methods such as NGT-QG would require much more efforts in order to make the combined method work competitively in the in-memory setting, which would be of independent interest. This is because in graph-based methods, the vectors to be searched are decided one after one based on the greedy search process in the run-time, and it is not easy to form batches of them so that SIMD-based fast implementation can be adopted. Therefore, we apply our method in combination with IVF index [45] in this paper. We leave it as future work to apply our quantization method in graph-based methods.

We present the workflow of the RaBitQ method with the IVF index as follows. During the index phase, for a set of raw data vectors, the IVF algorithm first clusters them with the KMeans algorithm, builds a bucket for each cluster and assigns the vectors to their corresponding buckets. Our method then normalizes the raw data vectors based on the centroids of their corresponding clusters and feeds the set of the normalized vectors to the subsequent steps of our RaBitQ method. During the query phase, for a raw query vector, the algorithm selects the first $N_{probe}$ clusters whose centroids are nearest to the query. Then for each selected cluster, the algorithm retrieves all the quantization codes and estimates their distances based on the quantization codes, which decide the vectors to be re-ranked based on exact distances.

As for re-ranking [84], PQ and its variants set a fixed hyper-parameter which decides the number of data vectors to be re-ranked (i.e., they re-rank the vectors with the smallest estimated distances). Specifically, they retrieve their raw data vectors, compute the exact distances and find out the final NN. In particular, the tuning of the hyper-parameter is empirical and often hard as it can vary largely across different datasets (see Section 5.2.3). In contrast, recall that in our method, there is a sharp error bound as discussed in Section 3.2 (note that the error bound is rigorous and always holds regardless of the data distribution). Thus, we decide the data vectors to be re-ranked based on the error bound without tuning hyper-parameters. Specifically, if a data vector has its lower bound of the distance greater than the exact distance of the currently searched nearest neighbor, then we drop it. Otherwise, we compute its exact distance for re-ranking. Due to the theoretical error bound, the re-ranking strategy has the guarantee of correctly sending the true NN from the probed clusters to re-ranking with high probability. The empirical verification can be found in Section 5.2.4. We emphasize that the idea of re-ranking based on a bound is not new. There are many studies from the database community that adopt a similar strategy [24, 27, 75, 88, 89, 93] for improving the robustness of similarity search for various data types. We note that beyond the idea of re-ranking based on an error bound, RaBitQ provides rigorous theoretical analysis on the tightness of the bounds and achieves the asymptotic optimality as we have discussed in Section 3.2.2.

Moreover, it is worth noting that re-ranking is a vital step for pushing forward RaBitQ's rigorous error bounds on the distances to the robustness of ANN search. In particular, when the ANN search requires higher accuracy than what RaBitQ can guarantee (e.g., when the true distances from the query to two different data vectors are extremely close to each other), then the estimated distance produced by RaBitQ would be less effective to rank them correctly. Re-ranking, in this case, is

necessary for achieving high recall. Note that it is inherently difficult for any methods of distance estimation when the distances are extremely close to each other.

## 5 EXPERIMENTS

### 5.1 Experimental Setup

Our experiments involve three folds. First, we compare our method with the conventional quantization methods in terms of the time-accuracy trade-off of distance estimation and time cost of index phase (with results shown in Section 5.2.1 and Section 5.2.2). Second, we compare the methods when applied for in-memory ANN (with results shown in Section 5.2.3). For ANN, we target to retrieve the 100 nearest neighbors for each query, i.e., $K = 100$, by following [30]. Third, we empirically verify our theoretical analysis (with results shown in Section 5.2.4 to 5.2.6). Finally, we note that RaBitQ is a method with rigorous theoretical guarantee. Its components are an integral whole and together explain its asymptotically optimal performance. The ablation of any component would cause the loss of the theoretical guarantee (i.e., the method becomes heuristic and the performance is no more theoretically predictable) and further disables the error-bound-based re-ranking (Section 4). Despite this, we include empirical ablation studies in the technical report [33].

**Datasets.** We use six public real-world datasets with varying sizes and dimensionalities, whose details can be found in Table 3. These datasets have been widely used to benchmark ANN algorithms [6, 58, 62, 67]. In particular, it has been reported that on the datasets SIFT, DEEP and GIST, PQx4fs and OPQx4fs have good empirical performance [5]. We note that all these public datasets provide both data and query vectors.

Table 3. Dataset Statistics

| Dataset | Size | $D$ | Query Size | Data Type |
|---|---|---|---|---|
| Msong | 992,272 | 420 | 200 | Audio |
| SIFT | 1,000,000 | 128 | 10,000 | Image |
| DEEP | 1,000,000 | 256 | 1,000 | Image |
| Word2Vec | 1,000,000 | 300 | 1,000 | Text |
| GIST | 1,000,000 | 960 | 1,000 | Image |
| Image | 2,340,373 | 150 | 200 | Image |

**Algorithms.** First, for estimating the distances between data vectors and query vectors, we consider three baselines, PQ [45], OPQ [34] and LSQ [66, 67]. In particular, **(1) PQ** and **(2) OPQ** are the most popular methods among the quantization methods [34, 45]. They are widely deployed in industry [48, 69, 83, 91]. The popularity of PQ and OPQ indicates that they have been empirically evaluated to the widest extent and are expected to have the best known stability. Thus, we adopt PQ and OPQ as the primary baseline methods representing the quantization methods which have no theoretical error bounds. There is another line of the quantization methods named the additive quantization [8, 66, 67, 94]. Compared with PQ, these methods target extreme accuracy at the cost of much higher time for optimizing the codebook and mapping the data vectors into quantization codes in the index phase. **(3) LSQ** [66, 67] is the state-of-the-art method of this line. Thus, we adopt LSQ as the baseline method representing the quantization methods which pursue extreme performance in the query phase. The baseline methods are taken from the 1.7.4 release version of the open-source library Faiss [48], which is well-optimized with the SIMD instructions of AVX2. Second, for ANN, we compare our method with the most competitive baseline method OPQ according to the results in Section 5.2.1. For both our method and OPQ, we combine them with the IVF index as specified in Section 4. We also include the comparison with **HNSW** [65] as a reference. It is one of the state-of-the-art graph-based methods as is benchmarked in [6, 85] and is also widely adopted

in industry [48, 69, 83, 91]. The implementation is taken from the hnswlib [65] optimized with the SIMD instructions of AVX2. We note that a recent quantization method ScaNN [37] proposes a new objective function for constructing the quantization codebook of PQ and claims better empirical performance. However, as has been reported [6], its superior performance is mainly due to the fast SIMD-based implementation [5]. The advantage vanishes when PQ is implemented with the same technique. Thus, we exclude it from the comparison. Furthermore, we exclude the comparison with the LSH methods because it has been reported that the quantization methods outperform these methods empirically by orders of magnitudes in efficiency when reaching the same recall [58]. The latest advances in LSH have not changed this trend [79]. Thus, comparable performance with the quantization methods indicates significant improvement over the LSH methods.

**Performance Metrics.** First, for estimating the distances between data vectors and query vectors, we use two metrics to measure the accuracy and one metric to measure the efficiency. In particular, we measure the accuracy with (1) the average relative error and (2) the maximum relative error on the estimated squared distances. The former measures the general quality of the estimated distances while the latter measures the robustness of the estimated distances. We measure the efficiency with the time for distance estimation per vector. We note that due to the effects of cache, the efficiency depends on the order of estimating distances for the vectors. To simulate the order when the methods are used in practice, we build the IVF index for all methods and estimate the distances in the order that the IVF index probes the clusters. We measure the end-to-end time of estimating distances for all the quantization codes in a dataset and divide it by the size of the dataset. We take the pre-processing time in the query phase (e.g., the time for normalizing, transforming and quantizing the query vector for our method) into account, thus making the comparisons fair. We also measure the time costs of the methods in the index phase. Second, for ANN, we adopt recall and average distance ratio for measuring the accuracy of ANN search. Specifically, recall is the ratio between the number of retrieved true nearest neighbors over $K$. Average distance ratio is the average of the distance ratios of the returned $K$ data vectors wrt the ground truth nearest neighbors. These metrics are widely adopted to measure the accuracy of ANN algorithms [6, 31, 38, 58, 77]. 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, 58, 85]. Following [6, 58, 85], 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.

**Parameter Setting.** As is suggested by Faiss [25], the number of clusters for IVF is set to be 4,096 as the datasets are at the million-scale. For our method, there are two parameters, i.e., $\epsilon_0$ and $B_q$. The theoretical analysis in Section 3.2.2 and Section 3.3.1 has provided clear suggestions that $\epsilon_0 = \Theta(\sqrt{\log(1/\delta)})$ and $B_q = \Theta(\log\log D)$, where $\delta$ is the failure probability. In practice, the parameters are fixed to be $\epsilon_0 = 1.9$ and $B_q = 4$ across all the datasets. The empirical parameter study can be found in Section 5.2.4 and Section 5.2.5. As for the length of the quantization code, it equals to $D$ by definition, but it can also be varied by padding the raw vectors with 0's before generating the quantization codes [7]. More padded 0's indicate longer quantization codes and higher accuracy due to Theorem 3.2 (recall that the error is bounded by $O(1/\sqrt{D})$). By default, the length of the quantization code is set to be the smallest multiple of 64 which is no smaller than $D$ (it is equal to or slightly larger than $D$) in order to make it possible to store the bit string with a sequence of 64-bit unsigned integers. For the conventional quantization methods (including PQ, OPQ and

[6]https://github.com/facebookresearch/faiss/wiki/Indexing-1M-vectors

[7]We emphasize that the padded dimensions will not be retained after the generation, and thus, will not affect the space and time costs related to the raw vectors.

LSQ), there are two parameters, namely the number of sub-segments of quantization codes $M$ and the number of candidates for re-ranking which should be tuned empirically. Following the default parameter setting [25, 37], we set the number of partitions to be $M = D/2$. We note that it cannot be further increased as $D$ should be divisible by $M$ for PQ and OPQ. The number of candidates for re-ranking is varied among 500, 1,000 and 2,500. The experimental results in Section 5.2.3 show that none of the parameters work consistently well across different datasets. For HNSW, we follow its original paper [65] by setting the number of maximum out-degree of each vertex in the graph as 32 (corresponding to $M_{HNSW} = 16$) and a parameter which controls the construction of the graph named $efConstruction$ as 500.

The C++ source codes are compiled by g++ 9.4.0 with `-Ofast -march=core-avx2` under Ubuntu 20.04LTS. The Python source codes are run on Python 3.8. All experiments are run on a machine with AMD Threadripper PRO 3955WX 3.9GHz processor (with Zen2 microarchitecture which supports the SIMD instructions till AVX2) and 64GB RAM. The code and datasets are available at https://github.com/gaoj0017/RaBitQ.

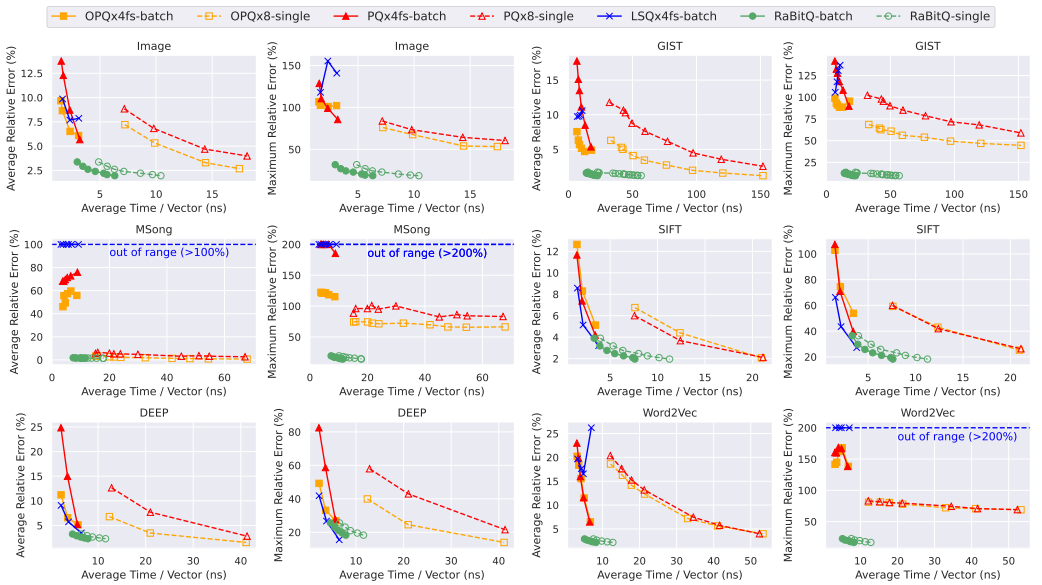


Fig. 3. Time-Accuracy Trade-Off for Distance Approximation. For baseline methods, (1) "x4fs-batch" means that the SIMD-based fast implementation is adopted (where 4 bits encode a quantized code and approximate distances for a batch of 32 data vectors are computed each time), and (2) "x8-single" means that 8 bits encode a quantized code and the approximate distance of one data vector is computed each time. In addition, the results of LSQx8-single are omitted since it, with the implementation from Faiss, has the time cost significantly larger than others.

## 5.2 Experimental Results

*5.2.1 Time-Accuracy Trade-Off per Vector for Distance Estimation.* We estimate the distance between a data vector (from the set of data vectors) and a query vector (from the set of query vectors) with different quantization methods including PQ, OPQ, LSQ and our RaBitQ method. We plot the "average relative error"-"time per vector" curve (left panels, bottom-left is better) and the "maximum relative error"-"time per vector" curve (right panels, bottom-left is better) by varying the length of the quantization codes in Figure 3. In particular, for our method, to plot the curve, we vary the

length by padding different number of 0's in the vectors when generating the quantization codes. For PQ, OPQ and LSQ, we vary the length by setting different $M$ (note that $D$ must be divisible by $M$ for PQ and OPQ).

Based on the results in Figure 3, we have the following observations. (1) LSQ has much less stable performance than PQ and OPQ. Except for the dataset SIFT and DEEP, LSQx4fs has its accuracy worse than PQx4fs and OPQx4fs. (2) Comparing the solid curves, we find that under the default setting of the number of bits (which corresponds to the last point in the red and orange solid curves and the first point in the green solid curve), our method shows consistently better accuracy than PQ and OPQ while having comparable efficiency on all the tested datasets. We emphasize that in the default setting, the length of the quantization code of our method is only around a half of those of PQ and OPQ (i.e., $D$ v.s. $2D$). (3) Comparing the dashed curves, we find that our method has significantly better efficiency than PQ and OPQ when reaching the same accuracy. (4) On the dataset Msong, PQx8 and OPQx8 have normal accuracy while PQx4fs and OPQx4fs have disastrous accuracy. It indicates that the reasonable accuracy of the conventional quantization methods with $k = 8$ does not indicate its normal performance with $k = 4$. Thus, it is not always feasible to speed up a conventional quantization method with the fast SIMD-based implementation [5]. On the other hand, the efficiency of the conventional quantization methods with $k = 8$ is hardly comparable with those with $k = 4$ on the other datasets. It indicates that the recent success of PQ in the in-memory ANN is largely attributed to the fast SIMD-based implementation. Thus, it is not a choice to replace the fast SIMD-based implementation with the original one in pursuit of the stability. (5) Except for the dataset SIFT and DEEP, PQx4fs and OPQx4fs have their maximum relative error of around 100%. It indicates that PQ and OPQ do not robustly produce high-accuracy estimated distances even on the datasets they perform well in general. As a comparison, our method has its maximum relative error at most 40% on all the tested datasets.

Table 4. The Indexing Time for the GIST Dataset

| | RaBitQ | PQ | OPQ | LSQ |
|---|---|---|---|---|
| Time | 117s | 105s | 291s | time-out (>24 hours) |

*5.2.2 Time in the Indexing Phase.* In Table 4, we report the indexing time of the quantization methods ($k = 4$ for PQ, OPQ and LSQ) under the default parameter setting on the GIST dataset with 32 threads on CPU. The results show that the indexing time is not a bottleneck for our method, PQ and OPQ since all of them can finish the indexing phase within a few mins. However, for LSQ, it takes more than 24 hours. This is because in LSQ, the step of mapping a data vector to its quantization code is NP-Hard [66, 67]. Although several techniques have been proposed for approximately solving the NP-Hard problem [66, 67], the time cost is still much larger than that of PQ, which largely limits its usage in practice.

*5.2.3 Time-Accuracy Trade-Off for ANN Search.* We then measure the performance of the algorithms when they are used in combination with the IVF index for ANN search. Considering the results in Section 5.2.1, we only include OPQx4fs-batch and RaBitQ-batch for the comparison as other methods or implementations are in general dominated when the quantization codes are allowed to be packed in batch. As a reference, we also include HNSW for comparison. We then plot the "QPS"-"recall" curve (left panel, upper-right is better) and the "QPS"-"average distance ratio" curve (right panel, upper-left is better) by varying the number of buckets to probe in the IVF index for the quantization methods in Figure 4. The curves for HNSW are plotted by varying a parameter named $efSearch$ which controls the QPS-recall tradeoff of HNSW. For OPQ, we show three curves which correspond to three different numbers of candidates for re-ranking. Based on Figure 4, we have the following observations. (1) On all the tested datasets, our method has consistently better

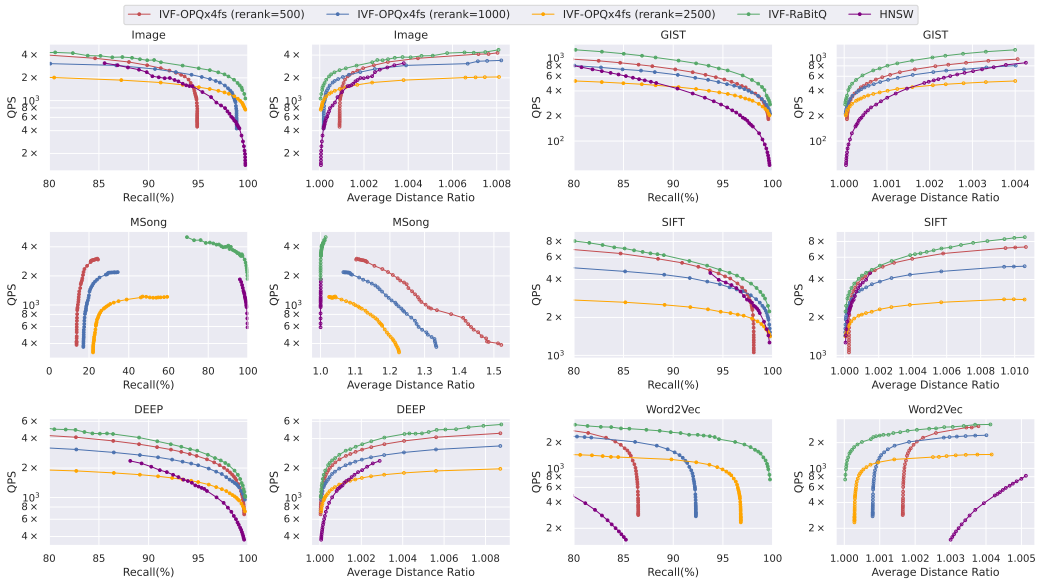


Fig. 4. Time-Accuracy Trade-Off for ANN Search. The parameter *rerank* represents the number of candidates for re-ranking.

performance than OPQ regardless of the re-ranking parameter. We emphasize that it has been reported that on the datasets SIFT, DEEP and GIST, OPQx4fs has good empirical performance [5]. Our method also consistently outperforms HNSW on all the tested datasets. (2) On the dataset MSong, the performance of OPQ is disastrous even with re-ranking applied. In particular, as the IVF index probes more buckets, the recall abnormally decreases because OPQ introduces too much error on the estimated distances. The poor accuracy shown in Figure 3 can explain the disastrous failure. (3) No single re-ranking parameter for OPQ works well across all the datasets. On SIFT, DEEP and GIST, 1,000 of candidates for re-ranking suffice to produce a nearly perfect recall while on Image and Word2Vec, a larger number of candidates for re-ranking is needed. We note that the tuning of the re-ranking parameter is often exhaustive as the parameters are intertwined with many factors such as the datasets and the other parameters. Prior to the testing, there is no reliable way to predict the optimal setting of parameters in practice. In contrast, recall that as is discussed in Section 3.2.2 and Section 3.3.1, in our method, the theoretical analysis provides explicit suggestions on the parameters. Thus, our method requires no tuning.

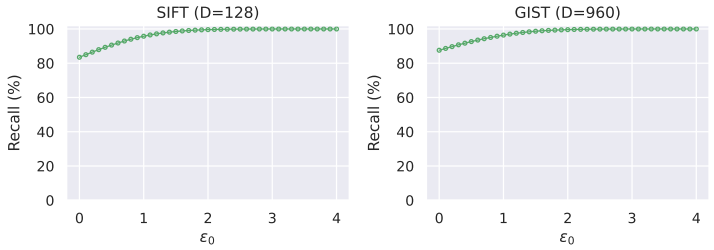


Fig. 5. Verification Study on $\epsilon_0$.

*5.2.4 Results for Verifying the Statement about $\epsilon_0$.* $\epsilon_0$ is a parameter which controls the confidence interval of the error bound (see Section 3.2.2). When the RaBitQ method is applied in ANN search,

it further controls the probability that we correctly send the NN to re-ranking (see Section 4). In particular, to make sure the failure probability be no greater than $\delta$, the theoretical analysis in Section 3.2.2 suggests to set $\epsilon_0 = \Theta(\sqrt{\log(1/\delta)})$. We emphasize that the statement is independent of any other factors such as the datasets or the setting of other parameters. This is the reason that the parameter needs no tuning. In Figure 5, we provide the empirical verification on the statement. In particular, we plot the "recall"-"$\epsilon_0$" curve by varying $\epsilon_0$ from 0.0 to 4.0. The recall is measured by estimating the distances for *all* the data vectors and decide the vectors to be re-ranked based on the strategy in Section 4 (note that if a true nearest neighbor is not re-ranked, it will be missed). Thus, the factors (other than the error of quantization methods) which may affect the recall are eliminated. Figure 5 shows that on two different datasets, both curves show highly similar trends that it achieves nearly perfect recall at around $\epsilon_0 = 1.9$.

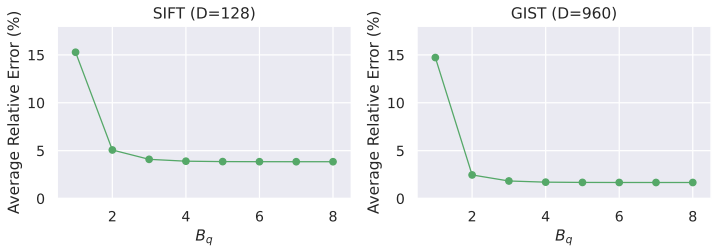


Fig. 6. Verification Study on $B_q$.

*5.2.5 Results for Verifying the Statement about $B_q$.* $B_q$ is a parameter which controls the error introduced in the computation of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$. Due to our analysis in Section 3.3.1, $B_q = \Theta(\log \log D)$ suffices to make sure that the error introduced in the computation of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ is much smaller than the error of the estimator. We note that $\Theta(\log \log D)$ varies extremely slowly with respect to $D$, and thus, it can be viewed as a constant when the dimensionality does not vary largely. In Figure 6, we provide the empirical verification on the statement. In particular, we plot the "average relative error"-"$B_q$" curve by varying $B_q$ from 1 to 8. Figure 6 shows that on two different datasets, both curves show highly similar trends that the error converges quickly at around $B_q = 4$. On the other hand, we would also like to highlight that further reducing $B_q$ would produce unignorable error in the computation of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$. In particular, when $B_q = 1$, i.e., both query and data vectors are quantized into binary strings, the error is much larger than the error when $B_q = 4$. This result may help to explain why the binary hashing methods cannot achieve good empirical performance.

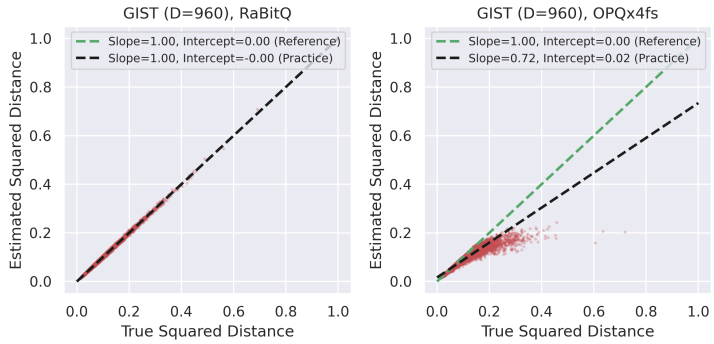


Fig. 7. Verification Study for Unbiasedness.

*5.2.6 Results for Verifying the Unbiasedness.* In Figure 7, we verify the unbiasedness of our method and show the biasedness of OPQ. We collect $10^7$ pairs of the estimated squared distances and the true squared distances between the query and data vectors (i.e., the first 10 query vectors in the query set and the $10^6$ data vectors in the full dataset of GIST) to verify the unbiasedness. The values of the distances are normalized by the maximum true squared distances. We fit the $10^7$ pairs with linear regression and plot the result with the black dashed line. We note that if a method is unbiased, the result of the linear regression should have the slope of 1 and the y-axis intercept of 0 (the green dashed line as a reference). Figure 7 clearly shows that our method is unbiased, which verifies the theoretical analysis in Section 3.2.2. On the other hand, the estimated distances produced by OPQ is clearly biased.

## 6 RELATED WORK

**Approximate Nearest Neighbor Search.** Existing studies on ANN search are usually categorized into four types: (1) graph-based methods [29, 30, 58, 64, 65, 73], (2) quantization-based methods [8, 34, 36, 37, 45, 66, 67, 72, 94], (3) tree-based methods [10, 15, 17, 70] and (4) hashing-based methods [18, 31, 35, 38, 39, 54, 56, 63, 77–79, 96]. We refer readers to recent tutorials [23, 74] and benchmarks/surveys [6, 7, 19, 58, 85, 87] for a comprehensive review. We note that recently, many studies design algorithms or systems by jointly considering different types of methods so that a method can enjoy the merits of both sides [1, 12, 13, 21, 43, 62, 95]. Our work proposes a quantization method which provides a sharp error bound and good empirical performance at the same time. Just like the conventional quantization methods, it can work as a component in an integrated algorithm or system. Our method has two additional advantages: (1) it involves no parameter tuning and (2) it supports efficient distance estimation for a single quantization code. These advantages may further smoothen its combination with other types of methods. Recently, there are a thread of methods which apply machine learning (ML) on ANN [9, 20, 26, 55, 57, 86, 90].

**Quantization.** There is a vast literature about the quantization of high-dimensional vectors from different communities including machine learning, computer vision and data management [8, 27, 34, 36, 45, 66, 67, 72, 80, 89, 94]. We refer readers to comprehensive surveys [24, 68, 82, 84] and reference books [75, 93]. It is worth of mentioning that in [80], a quantization method called Split-VQ was mentioned. The method covers the major idea of PQ, i.e., it splits the vectors into sub-segments, constructs sub-codebooks for each sub-segment and forms the codebook with Cartesian product. Besides PQ and its variants, there are other types of quantization methods, e.g., scalar quantization [1, 27, 89]. These methods quantize the scalar values of each dimension separately, which often adopt more moderate compression rates than PQ in exchange for better accuracy. In particular, VA+ File [27], a scalar quantization method, has shown leading performance on the similarity search of data series according to a recent benchmark [24]. Besides the studies on the quantization algorithms, we note that hardware-aware optimization (with SIMD, GPU, FPGA, etc) also makes significant contributions to the performance of these methods [4, 5, 47, 48, 61]. To inherit the merits of the well-developed hardware-aware optimization, in this work, we reduce our computation to the computation of PQ (Section 3.3). However, RaBitQ, in its nature, can be implemented with much simpler bitwise operations (which is not possible for PQ and its variants). It remains to be an interesting question whether dedicated hardware-aware optimization can further improve the performance of RaBitQ.

**Theoretical Studies on High-Dimensional Vectors.** The theoretical studies on high-dimensional vectors are primarily about the seminal Johnson-Lindenstrauss (JL) Lemma [49]. It presents that reducing the dimensionality of a vector to $O(\epsilon^{-2}\log(1/\delta))$ suffices to guarantee the error bound of $\epsilon$. Recent advances improve the JL Lemma in different aspects. For example, [53] proves the optimality

of the JL Lemma. [2, 50] propose fast algorithms to do the dimension reduction. We refer readers to a comprehensive survey [28]. Our method fits into a recent line of studies [3, 40, 41, 71], which target to improve the JL Lemma by compressing high-dimensional vectors into short codes. As a comparison, to guarantee an error bound of $\epsilon$, the JL Lemma requires a vector of $O(\epsilon^{-2}\log(1/\delta))$ *dimensions* while these studies prove that a short code with $O(\epsilon^{-2}\log(1/\delta))$ *bits* would be sufficient. In practical terms, we note that although the existing studies achieve the improvement in theory in terms of the space complexity (i.e., the minimum number of bits needed for guaranteeing a certain error bound), they care less about the improvement in efficiency. In particular, these methods do not suit the in-memory ANN search because, for estimating the distance during the query phase, they need decompress the short codes and compute the distances with the decompressed vectors, which degrades to the brute force in efficiency. For this reason, these methods have not been adopted in practice. In contrast, our method supports practically efficient computation as is specified in Section 3.3.

**Signed Random Projection.** We note that there are a line of studies named signed random projection (SRP) which generate a short code for estimating the angular values between vectors via binarizing the vectors after randomization [11, 22, 46, 51]. We note that our method is different from these studies in the following aspects. (1) Problem-wise, SRP targets to unbiasedly estimate the angular value while RaBitQ targets to unbiasedly estimate the inner product (and further, the squared distances). Note that the relationship between the angular value and the inner product is non-linear. The unbiased estimator for one does not trivially derive an unbiased estimator for the other. (2) Theory-wise, RaBitQ has a stronger type of guarantee than SRP. In particular, RaBitQ guarantees that every data vector has its distance within the bounds with high probability. In contrast, SRP only bounds the variance, i.e., the "average" squared error, and it does not provide a bound for every estimated value. Thus, it cannot help with the re-ranking in similarity search. (3) Technique-wise, in SRP the bit strings are viewed as some hashing codes while in RaBitQ, the bit strings are the binary representations of bi-valued vectors. Moreover, SRP maps both the data and query vectors to bit strings, which introduces error from both sides. In contrast, RaBitQ quantizes the data vectors to be bit strings and the query vectors to be vectors of 4-bit unsigned integers. Theorem 3.3 proves that quantizing the query vectors only introduces negligible error. Thus, RaBitQ only introduces the error from the side of the data vector.

## 7 CONCLUSION

In conclusion, we propose a novel randomized quantization method RaBitQ which has clear advantages in both empirical accuracy and rigorous theoretical error bound over PQ and its variants. The proposed efficient implementations based on simple bitwise operations or fast SIMD-based operations further make it stand out in terms of the time-accuracy trade-off for the in-memory ANN search. Extensive experiments on real-world datasets verify both (1) the empirical superiority of our method in terms of the time-accuracy trade-off and (2) the alignment of the empirical performance with the theoretical analysis. Some interesting research directions include applying our method in other scenarios of ANN search (e.g., with graph-based indexes or on other storage devices [14, 42, 44, 59]). Besides, RaBitQ can also be trivially applied to unbiasedly estimate cosine similarity and inner product [8], which further implies its potential in maximum inner product search and neural network quantization.

[8]The cosine similarity of two vectors exactly equals to the inner product of their unit vectors. The inner product of $\mathbf{o}$ and $\mathbf{q}$ can be expressed as $\langle \mathbf{o}, \mathbf{q}\rangle = \langle \mathbf{o}-\mathbf{c}+\mathbf{c}, \mathbf{q}-\mathbf{c}+\mathbf{c}\rangle = \|\mathbf{o}-\mathbf{c}\| \cdot \|\mathbf{q}-\mathbf{c}\| \cdot \langle (\mathbf{o}-\mathbf{c})/\|\mathbf{o}-\mathbf{c}\|, (\mathbf{q}-\mathbf{c})/\|\mathbf{q}-\mathbf{c}\|\rangle + \langle \mathbf{o}, \mathbf{c}\rangle + \langle \mathbf{q}, \mathbf{c}\rangle - \|\mathbf{c}\|^2$, where $\mathbf{c}$ is the centroid of the data vectors, and it reduces to the estimation of inner product between unit vectors as we do in Section 3.1.1.

## ACKNOWLEDGEMENTS

We would like to thank 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, Tier 2 Award MOE-T2EP20220-0011, and Tier 1 Award (RG77/21)). 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. https://doi.org/10.14778/3611479.3611537

[2] Nir Ailon and Bernard Chazelle. 2009. The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors. *SIAM J. Comput.* 39, 1 (2009), 302–322. https://doi.org/10.1137/060673096 arXiv:https://doi.org/10.1137/060673096

[3] 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. https://doi.org/10.1109/FOCS.2017.65

[4] 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. https://doi.org/10.14778/2856318.2856324

[5] 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. https://doi.org/10.1145/3078971.3078992

[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. https://doi.org/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. https://doi.org/10.1109/CVPR.2014.124

[9] Dmitry Baranchuk, Dmitry Persiyanov, Anton Sinitsin, and Artem Babenko. 2019. Learning to Route in Similarity Graphs. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 475–484. https://proceedings.mlr.press/v97/baranchuk19a.html

[10] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*. 97–104.

[11] 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. https://doi.org/10.1145/509907.509965

[12] Patrick H. Chen, Wei-Cheng Chang, Jyun-Yu Jiang, Hsiang-Fu Yu, Inderjit S. Dhillon, and Cho-Jui Hsieh. 2023. FINGER: Fast inference for graph-based approximate nearest neighbor search. In *The Web Conference 2023*. https://www.amazon.science/publications/finger-fast-inference-for-graph-based-approximate-nearest-neighbor-search

[13] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search.* https://github.com/Microsoft/SPTAG

[14] 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)*.

[15] 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.

[16] T. Cover and P. Hart. 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13, 1 (1967), 21–27. https://doi.org/10.1109/TIT.1967.1053964

[17] 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.

[18] 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.
[19] 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).
[20] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. 2020. Learning Space Partitions for Nearest Neighbor Search. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rkenmREFDr
[21] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. 2018. Link and Code: Fast Indexing with Graphs and Compact Regression Codes. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3646–3654. https://doi.org/10.1109/CVPR.2018.00384
[22] Punit Pankaj Dubey, Bhisham Dev Verma, Rameshwar Pratap, and Keegan Kang. 2022. Improving sign-random-projection via count sketch. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence (Proceedings of Machine Learning Research, Vol. 180)*, James Cussens and Kun Zhang (Eds.). PMLR, 599–609. https://proceedings.mlr.press/v180/dubey22a.html
[23] 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. https://doi.org/10.14778/3476311.3476407
[24] 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. https://doi.org/10.14778/3282495.3282498
[25] Faiss. 2023. Faiss. https://github.com/facebookresearch/faiss.
[26] Chao Feng, Defu Lian, Xiting Wang, Zheng Liu, Xing Xie, and Enhong Chen. 2022. Reinforcement Routing on Proximity Graph for Efficient Recommendation. *ACM Trans. Inf. Syst.* (jan 2022). https://doi.org/10.1145/3512767 Just Accepted.
[27] 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. https://doi.org/10.1145/354756.354820
[28] Casper Benjamin Freksen. 2021. An Introduction to Johnson-Lindenstrauss Transforms. *CoRR* abs/2103.00564 (2021). arXiv:2103.00564 https://arxiv.org/abs/2103.00564
[29] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
[30] 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. https://doi.org/10.14778/3303753.3303754
[31] 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. https://doi.org/10.1145/2213836.2213898
[32] 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. https://doi.org/10.1145/3589282
[33] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with Theoretical Error Bound for Approximate Nearest Neighbor Search (Technical Report). https://github.com/gaoj0017/RaBitQ/technical_report.pdf.
[34] 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.
[35] Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. 2020. IDEC: Indexable Distance Estimating Codes for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 13, 9 (may 2020), 1483–1497. https://doi.org/10.14778/3397230.3397243
[36] 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. https://doi.org/10.1109/TPAMI.2012.193
[37] 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.
[38] 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.
[39] 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.

[40] Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. 2017. Practical Data-Dependent Metric Compression with Provable Guarantees. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 2614–2623.

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

[42] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 2023. CXL-ANNS: Software-Hardware Collaborative Memory Disaggregation and Computation for Billion-Scale Approximate Nearest Neighbor Search. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 585–600. https://www.usenix.org/conference/atc23/presentation/jang

[43] Yahoo Japan. 2022. NGT-QG. https://github.com/yahoojapan/NGT.

[44] 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

[45] 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.

[46] Jianqiu Ji, Jianmin Li, Shuicheng Yan, Bo Zhang, and Qi Tian. 2012. Super-Bit Locality-Sensitive Hashing. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (Lake Tahoe, Nevada) *(NIPS'12)*. Curran Associates Inc., Red Hook, NY, USA, 108–116.

[47] 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. https://doi.org/10.1145/3581784.3607045

[48] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

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

[50] Daniel M. Kane and Jelani Nelson. 2014. Sparser Johnson-Lindenstrauss Transforms. *J. ACM* 61, 1, Article 4 (jan 2014), 23 pages. https://doi.org/10.1145/2559902

[51] Keegan Kang and Weipin Wong. 2018. Improving Sign Random Projections With Additional Information. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 2479–2487. https://proceedings.mlr.press/v80/kang18b.html

[52] V. I. Khokhlov. 2006. The Uniform Distribution on a Sphere in $\mathbf{R}^S$. Properties of Projections. I. *Theory of Probability & Its Applications* 50, 3 (2006), 386–399. https://doi.org/10.1137/S0040585X97981846 arXiv:https://doi.org/10.1137/S0040585X97981846

[53] 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.

[54] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony K. H. Tung. 2020. Locality-Sensitive Hashing Scheme Based on Longest Circular Co-Substring. 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, 2589–2599. https://doi.org/10.1145/3318464.3389778

[55] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. 2020. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA). Association for Computing Machinery, New York, NY, USA, 2539–2554. https://doi.org/10.1145/3318464.3380600

[56] Jinfeng Li, Xiao Yan, Jian Zhang, An Xu, James Cheng, Jie Liu, Kelvin K. W. Ng, and Ti-chung Cheng. 2018. A General and Efficient Querying Method for Learning to Hash. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1333–1347. https://doi.org/10.1145/3183713.3183750

[57] Mingjie Li, Yuan-Gen Wang, Peng Zhang, Hanpin Wang, Lisheng Fan, Enxia Li, and Wei Wang. 2023. Deep Learning for Approximate Nearest Neighbour Search: A Survey and Future Directions. *IEEE Transactions on Knowledge and Data Engineering* 35, 9 (2023), 8997–9018. https://doi.org/10.1109/TKDE.2022.3220683

[58] 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.

[59] Yingfan Liu, Hong Cheng, and Jiangtao Cui. 2017. PQBF: I/O-Efficient Approximate Nearest Neighbor Search by Product Quantization. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (Singapore, Singapore) *(CIKM '17)*. Association for Computing Machinery, New York, NY, USA, 667–676. https://doi.org/10.1145/3132847.3132901

[60] Ying Liu, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. 2007. A survey of content-based image retrieval with high-level semantics. *Pattern Recognition* 40, 1 (2007), 262–282. https://doi.org/10.1016/j.patcog.2006.04.045

[61] Zihan Liu, Wentao Ni, Jingwen Leng, Yu Feng, Cong Guo, Quan Chen, Chao Li, Minyi Guo, and Yuhao Zhu. 2023. JUNO: Optimizing High-Dimensional Approximate Nearest Neighbour Search with Sparsity-Aware Algorithm and Ray-Tracing Core Mapping. arXiv:2312.01712 [cs.DC]

[62] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: Hierarchical Graph Structure Based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 15, 2 (oct 2021), 246–258. https://doi.org/10.14778/3489496.3489506

[63] Kejing Lu, Hongya Wang, Wei Wang, and Mineichi Kudo. 2020. VHP: approximate nearest neighbor search via virtual hypersphere partitioning. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1443–1455.

[64] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68. https://doi.org/10.1016/j.is.2013.10.006

[65] 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. https://doi.org/10.1109/TPAMI.2018.2889473

[66] 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.

[67] 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. https://doi.org/10.1007/978-3-030-01270-0_30

[68] 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.

[69] 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. https://doi.org/10.1145/3589777

[70] 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.

[71] Rasmus Pagh and Johan Sivertsen. 2020. The Space Complexity of Inner Product Filters. In *23rd International Conference on Database Theory (ICDT 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 155)*, Carsten Lutz and Jean Christoph Jung (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 22:1–22:14. https://doi.org/10.4230/LIPIcs.ICDT.2020.22

[72] 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. https://doi.org/10.1109/ICDE53745.2022.00268

[73] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-Dimensional Databases. *Proc. ACM Manag. Data* 1, 1, Article 54 (may 2023), 27 pages. https://doi.org/10.1145/3588908

[74] 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. https://doi.org/10.1145/3447548.3470811

[75] 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.

[76] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. *Collaborative Filtering Recommender Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 291–324. https://doi.org/10.1007/978-3-540-72079-9_9

[77] 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).

[78] 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.

[79] Y. Tian, X. Zhao, and X. Zhou. 2022. DB-LSH: Locality-Sensitive Hashing with Query-based Dynamic Bucketing. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 2250–2262. https://doi.org/10.1109/ICDE53745.2022.00214

[80] 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. https://doi.org/10.1145/641007.641117

[81] Roman Vershynin. 2018. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge University Press. https://doi.org/10.1017/9781108231596

[82] 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. https://doi.org/10.1109/JPROC.2015.2487976

[83] 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. https://doi.org/10.1145/3448016.3457550

[84] 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. https://doi.org/10.1109/TPAMI.2017.2699960

[85] 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. https://doi.org/10.14778/3476249.3476255

[86] Yifan Wang, Haodi Ma, and Daisy Zhe Wang. 2022. LIDER: An Efficient High-Dimensional Learned Index for Large-Scale Dense Passage Retrieval. *Proc. VLDB Endow.* 16, 2 (oct 2022), 154–166. https://doi.org/10.14778/3565816.3565819

[87] 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.

[88] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A compact and adaptive index for large data series collections. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[89] 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.

[90] Shitao Xiao, Zheng Liu, Weihao Han, Jianjin Zhang, Defu Lian, Yeyun Gong, Qi Chen, Fan Yang, Hao Sun, Yingxia Shao, and Xing Xie. 2022. Distill-VQ: Learning Retrieval Oriented Vector Quantization By Distilling Knowledge from Dense Embeddings. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Madrid, Spain) *(SIGIR '22)*. ACM, New York, NY, USA, 1513–1523. https://doi.org/10.1145/3477495.3531799

[91] 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. https://doi.org/10.1145/3318464.3386131

[92] R. Zamir and M. Feder. 1992. On universal quantization by randomized uniform/lattice quantizers. *IEEE Transactions on Information Theory* 38, 2 (1992), 428–436. https://doi.org/10.1109/18.119699

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

[94] 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

[95] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *Proc. VLDB Endow.* 16, 8 (jun 2023), 1979–1991. https://doi.org/10.14778/3594512.3594527

[96] Bolong Zheng, Zhao Xi, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S Jensen. 2020. PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search. *Proceedings of the VLDB Endowment* 13, 5 (2020), 643–655.

Received October 2023; revised January 2024; accepted February 2024