High-Dimensional Vector Quantization: General Framework, Recent Advances, and Future Directions

Jianyang Gao†, Yutong Gou†, Yuexuan Xu†, Jifan Shi†, Cheng Long†,*
Raymond Chi-Wing Wong§, Themis Palpanas‡
†Nanyang Technological University
jianyang.gao, c.long@ntu.edu.sg, yutong003, yuexuan001, jifan002@e.ntu.edu.sg
§The Hong Kong University of Science and Technology
raywong@cse.ust.hk
‡LIPADE, Université Paris Cité
themis@mi.parisdescartes.fr

Abstract

Vector quantization is fundamental for high-dimensional vector data management. In this paper, we first introduce the background of vector data management and vector quantization. We then present the general framework of vector quantization and discuss existing popular schemes. We next introduce recent advances of quantization, namely the RaBitQ method, which provides optimized approaches for binary and scalar quantization and achieves asymptotic optimality. In particular, we discuss RaBitQ's insights, codebooks and distance estimator. Finally, we discuss opportunities of building data-aware quantization methods with theoretical error bounds for the future work.

1 Introduction

With the explosive growth of unstructured data, including images, text, and audio, the need for effective processing and management of the data has become increasingly urgent. Recent advances in deep learning have enabled the extraction of semantic information from unstructured data through deep neural network models such as GPT, DeepSeek, and Llama. These models can represent unstructured data as high-dimensional vectors, which can then be utilized in downstream models for various applications, including classification and generation. Additionally, these vector representations can be stored in database management systems to support semantic-based management and retrieval of unstructured data, with applications across diverse fields including databases [15], information retrieval [48, 49], recommendation [50], and retrieval-augmented generation (RAG) [51].

The nearest neighbor (NN) query is a fundamental operation in vector data management [4, 13, 14, 17, 22, 30–36, 53–56, 61, 63–65]. Formally, given a database of vectors, an NN query receives a query vector and aims to find the closest data vector from the query vector in the database. However, due to the curse of dimensionality, it has been proven that designing algorithms for exact NN queries is intrinsically difficult, with no algorithm able to achieve sub-linear worst-case time complexity [52]. To address this challenge, researchers often relax the problem to an approximate nearest neighbor (ANN) query for better time-accuracy trade-off. Despite the relaxation, existing methods [17, 53–56] still suffer from high time consumption for vectors generated by deep neural network models due to their high dimensionality. For instance, in early 2024, OpenAI's most powerful model, text-embedding-3-large[1],

*Cheng Long is the corresponding author.

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

generated vectors with 1,536 or 3,072 dimensions. For these vectors, the running time of most in-memory ANN algorithms is dominated by that of distance computations between vectors, since each computation requires thousands of arithmetic operations on floating-point numbers [8]. The space consumption is also dominated by that of storing the vectors due to their high dimensionality.

Vector quantization addresses this challenge by reducing the precision of vector representations while preserving their essential characteristics. Specifically, they construct a codebook conceptually, map each data vector to its most similar codeword within the codebook, and store the corresponding short code to achieve space efficiency. During querying, distances between data vectors and query vectors can be estimated solely based on the compressed codes, which is more efficient than computing the distances based on raw vectors. Many real-world vector database systems and libraries, e.g., Milvus and FAISS, deploy vector quantization to speed up and save space for ANN queries over high-dimensional vector data [15, 20, 21, 45, 46]. Quantization also plays an important role in neural network quantization/compression for efficient inference and/or deployment under resource-constrained settings [11, 12].

There exists a substantial body of literature on vector quantization across various fields, including machine learning, computer vision, and data management [1, 7, 9, 10, 16–18, 23, 28]. Scalar quantization (SQ) and its variants [1, 57, 60] take finite uniform grids as codebooks and map a floating-point vector to a vector of unsigned integers, enabling distance estimation through arithmetic operations on unsigned integers without the need for decompression. Product quantization (PQ) adopts machine learning techniques, such as clustering, in codebook construction [17, 18]. Additive quantization (AQ) further increases the flexibility of learning, by enlarging the search space of codebooks [7, 16, 23]. Recent studies utilize neural networks to construct codebooks, enabling an even larger search space of codebooks [38, 39]. Despite the increased flexibility of codebooks, these learning-based methods degenerate the efficiency of distance estimation and vector quantization, and cause the loss of theoretical error bounds.

More recently, a new quantization method called RaBitQ has been proposed [9, 10], offering optimized approaches for binary quantization and scalar quantization. It achieves asymptotic optimality on the space-accuracy trade-off, and guarantees unbiased distance estimation. RaBitQ consistently outperforms existing popular methods in real-world systems including PQ, SQ and their variants. Given its promising performance, RaBitQ is attracting much attention from the industry and has been adopted in several real-world systems in industry recently. For example, TensorChord has developed a highly cost-efficient vector search solution, with RaBitQ serving as one of its key components[2]. ElasticSearch adopts RaBitQ with some minor modifications for vector quantization[3]. In particular, RaBitQ strikes the right balance in the space-accuracy trade-off by leveraging the concentration of measure [24], a phenomenon that exhibits itself in high-dimensional spaces. This unique property in high-dimensional spaces allows the algorithm to accurately estimate certain variables without doing any computations, which brings performance gains in accuracy, at no cost. Moreover, RaBitQ's distance estimation can be efficiently supported by bitwise operations, and arithmetic operations of unsigned integers, since it adopts transformed finite uniform grids as its codebook, i.e., a codebook of binary and scalar quantization. For this codebook, it designs an algorithm for finding the best-match codeword. In particular, to find better quantized vector beyond scalar quantization, it tries different re-scaling parameters on a per-vector basis. For each re-scaling parameter, it performs scalar quantization on the rescaled vectors and computes the similarity between the original vector and the quantized vector. After trying various parameters, it finds the optimal rescaling parameter and the best-match codeword which minimizes the quantization error.

In this paper, we first study the general framework of vector quantization and discuss how existing studies fit into the framework. In addition, we discuss how some existing methods incorporate learning

[2] https://blog.vectorchord.ai/vectorchord-store-400k-vectors-for-1-in-postgresql

[3] https://github.com/apache/lucene/pull/13651

into the framework. We then present intuitive explanations on the state-of-the-art quantization method called RaBitQ [9, 10]. The method offers optimized approaches for binary and scalar quantization, provides unbiased distance estimation, and achieves the asymptotically optimal error bound. Finally, we discuss a promising research direction: how to design a data-aware quantization method with theoretical error bounds.

2 General Framework and Popular Schemes of Quantization

In this section, we first illustrate the general framework of vector quantization. Then, we discuss existing schemes of quantization which are applied for high-dimensional vector data management, particularly, nearest neighbor search. Finally, we introduce several other vector compression schemes.

2.1 The General Framework of Vector Quantization

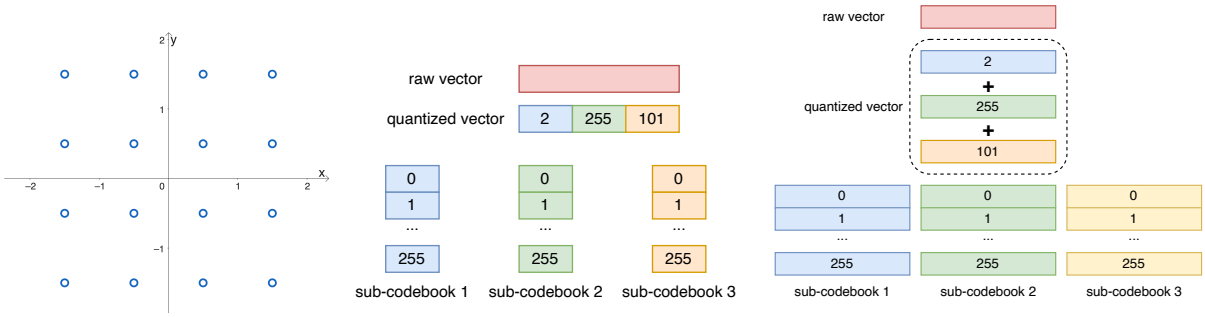
Vector quantization is a longstanding research topic [7, 9, 10, 16–18, 23, 28, 29, 62]. The general functionality of quantization is to represent continuous data objects by discrete codes of finite length. It is also used for lossy data compression, i.e., compressing high-resolution discrete data objects into codes with fewer bits. In particular, in high-dimensional vector data management, letting D be the dimensionality, quantization is used to represent vectors in $\mathbb{R}^D$ by codes of certain length. The code is subsequently used to perform computational tasks. In this paper, all algorithms can be used to estimate metrics in Euclidean spaces including Euclidean distances, inner products and cosine similarities. To simplify the presentation, we describe quantization algorithms using Euclidean distances as an example. The cases for inner products and cosine similarities can be addressed similarly. Furthermore, in approximate nearest neighbor (ANN) search, distance estimation is usually performed on a data vector $\mathbf{o}_r$ and a query vector $\mathbf{q}_r$. We focus on estimating the squared distances $\|\mathbf{o}_r - \mathbf{q}_r\|^2$. A vector quantization algorithm involves the following two critical components.

- **Codebook**: A *codebook* $\mathcal{C}$ refers to a finite set of vectors in $\mathbb{R}^d$. The elements in a codebook are referred to as *codewords*. Each codeword has a unique representation, namely a *code*, which can be physically stored in computers. An algorithm quantizes a vector by finding the nearest codeword from a codebook as its quantized vector. The corresponding code of the quantized vector is then stored.
- **Distance estimator**: A distance estimator is a function based on the query vector and the quantized data vector of a raw vector, which aims to estimte the distance between the query vector and the raw vector. It can be computed during querying without retrieving the raw vectors.

For a quantization algorithm, we primarily focus on its performance in the following aspects: (1) space-accuracy trade-off (i.e., the trade-off between the length of a quantized code and accuracy of the estimated distance based on the quantized code), (2) distance estimation time and (3) vector quantization time. The performance of a quantization algorithm is largely determined by the codebook construction and distance estimator, which we explain with existing schemes of methods.

2.2 Scalar Quantization and Binary Quantization

Scalar quantization (SQ) is a family of classical algorithms which are widely deployed in real-world systems [15, 20, 21]. The methods take finite uniform grids, i.e., the vectors whose coordinates are B-bit unsigned integers, and resize them to form the codebook an illustration is shown in Figure 1a. Binary quantization is a special case of SQ with $B = 1$. Let v_l and v_r be a lower bound and an upper bound



(a) Scalar Quantization (b) Product Quantization (c) Additive Quantization

Figure 1: The illustrations of codebooks. The codebooks of product quantization and additive quantization do not have a clear geometric illustration as that of the scalar quantization does.

for resizing the codebook respectively (i.e., the length of the grid is $\frac{v_r - v_l}{2^B - 1}$). The classical SQ takes the minimum and maximum values of a dataset for v_l and v_r. The state-of-the-art variant LVQ [1] takes the minimum and maximum values of the coordinates of every individual vector as v_l and v_r, respectively. Let $\Delta := \frac{v_r - v_l}{2^B - 1}$ be the length of a grid cell. The codebook of SQ is illustrated in Figure 1a and is formally presented as follows.

$$\mathcal{C} = \left\{ \Delta \cdot \mathbf{c} + v_l \cdot \mathbf{1}_D \middle| \mathbf{c}[i] = 0, 1, 2, ..., 2^B - 1 \right\} \tag{1}$$

Here $\mathbf{1}_D$ denotes a D-dimensional vector whose coordinates are ones.

The simple codebook of SQ, i.e., finite uniform grids, leads to clear advantages in vector quantization time and distance estimation time. Specifically, the vector quantization can be performed by rounding every coordinate of a vector to its nearest boundary of the grid cells for the corresponding dimension. The distance estimation can be realized with arithmetic operations of unsigned integers, i.e., the computation can be achieved without decompressing the codes. It is worth highlighting that when $B = 1$, i.e., the case of binary quantization, the computation can be realized with efficient bitwise operations. Despite this, the classical SQ and LVQ have limitations in the space-accuracy trade-off using a small number of bits. As has been reported, these methods can hardly achieve reasonable accuracy when using 1-bit and 2-bit quantization, which correspond to 32x and 16x compression rates, respectively [10].

2.3 Product Quantization

Product quantization (PQ) is a popular thread of quantization methods [17–19, 58]. Unlike SQ, which uses finite uniform grids as the codebook, PQ incorporates learning into the codebook construction process. This characteristic makes PQ and its variants part of the broader category known as *learning to hash* [19]. It constructs its codebook by optimizing within a large search space, aiming to learn the underlying distribution of the dataset through this optimization process. Specifically, for PQ, it divides D dimensions of a vector into M sub-segments, with each sub-segment containing D/M dimensions. For each sub-segment, it constructs a sub-codebook by taking 2^k centroids (by default, $k = 8$) obtained through clustering, a typical unsupervised learning task on a dataset. Finally, PQ takes the Cartesian product of the sub-codebooks as the codebook of PQ. Let $\mathcal{C}_i$ be the i-th sub-codebook, the codebook of PQ is illustrated in Figure 1b and is formally presented as follows.

$$\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times ... \times \mathcal{C}_M \tag{2}$$

In this scheme, to quantize a vector, it can separately process each sub-segment. For each sub-segment, it can find the nearest codeword by computing the vector's distances from all codewords. OPQ further

introduces a learned rotation before the codebook construction [18]. VAQ proposes to non-uniformly assign bits to different subspaces according to their importance [58]. Both further enlarge the search space of codebooks in the learning process.

Introducing learning to the codebook construction leads to better flexibility and data-awareness, resulting in better space-accuracy trade-off under a large compression rate, e.g., 32x and 16x compression [10, 17, 18]. However, the codebook of PQ also causes significant degeneration on the efficiency of distance estimation. Specifically, during querying, they adopt asymmetric distance computation to estimate the distance [17]. When a query comes, it prepares a look-up-table (LUT) for every sub-codebook. The i-th LUT stores 2^k numbers which represent the squared distances between the codewords in the i-th sub-codebook and i-th sub-segment of the query vector. For a given code, by looking up and accumulating the values in the LUTs for M times, where M is the number of sub-segments, it can compute an estimated distance. As has been comprehensively studied [3], under the same compression rates, this computation is much less efficient compared with that of SQ, whose computation can be realized with bitwise operations or arithmetic operations of unsigned integers. The operation of looking up tables incurs frequent random memory accesses and would be slow when M is large, where the LUTs cannot be fully hosted in L1 cache [1]. To mitigate the issue, FastScan is proposed to estimate distances batch by batch based on SIMD (Single Instruction Multiple Data) instructions. It significantly accelerates the process and can be easily deployed in clustering-based indices such as IVF [3, 5, 6, 21, 59]. Despite this, there has been no known efficient implementation for estimating distances between two individual vectors, which largely limits the efficiency of PQ and its variants when they are combined with graph-based indices [1]. These studies underscore a fundamental trade-off between the flexibility of codebooks and the efficiency of distance estimation: integrating learning into a specific component of the quantization pipeline may not necessarily enhance overall performance in ANN queries. Furthermore, it is worth highlighting that although the search space of PQ's codebook covers the codebook of SQ, due to its heuristic objectives and approximate optimization algorithms, PQ does not necessarily find a better codebook and produce better accuracy than SQ. In fact, PQ is consistently worse than SQ and its variants when a moderate compression rate is used [10].

2.4 Additive Quantization

Additive quantization (AQ) further generalizes PQ by considering a larger search space of the codebook [7, 16, 23]. It considers M sub-codebooks, where each contains 2^k codewords. Unlike PQ whose sub-codebooks are formed of vectors in (D/M)-dimensional spaces, the sub-codebooks of AQ are formed of vectors in D-dimensional spaces. The codeword of AQ is formed of the summation of the codeword in the sub-codebooks. Let $\mathcal{C}_i$ be the i-th sub-codebook. The codebook of AQ is illustrated in Figure 1c and is formally presented as follows.

$$\mathcal{C} = \left\{ \sum_{i=1}^{M} \mathbf{c}_i \middle| \mathbf{c}_i \in \mathcal{C}_i \right\} \tag{3}$$

The search space of AQ's codebook is strictly a superset of that of PQ's, indicating that it has even better flexibility of learning [7, 16, 23]. This brings improvement of space-accuracy trade-off under aggressive compression rates [7, 16, 21, 23]. For instance, when quantizing a vector with 32 bits, 64 bits and 128 bits, these methods have shown better accuracy than PQ and its variants [16, 21]. However, the scheme of codebook construction also causes the intrinsic hardness of vector quantization and limits their scalability. Specifically, quantizing a vector requires enumerating all $2^{M\times k}$ codewords to exactly find the nearest one, which is computationally impractical. Existing methods all adopt approximate algorithms to mitigate this issue [7, 16, 23]. Despite this, the time costs of quantization are still significantly higher

than other schemes of methods. Additionally, similar to the comparison between SQ and PQ, although the search space of AQ's codebook covers the search space of PQ's, due to the approximation nature of their optimization algorithms, the AQ methods do not always find a better codebook and provide better accuracy than PQ [9], especially when the compression rate is moderate.

2.5 Other Schemes of Vector Compression

In recent years, there have been several studies, which construct codebook with neural networks [38, 39]. UNQ implicitly constructs its codebook via training a neural network consisting of an encoder and a decoder [38]. Specifically, for quantizing a vector, it encodes the raw vectors into a hidden space, quantizes the vectors in the hidden space into codes, and generates the quantized vector via decoding the codes. In this scheme, the encoder and decoder implicitly decide the codebook. QINCo [39] introduces neural networks into the workflow of residual quantization [37]. The introduction of neural networks further enhances the flexibility of codebook construction. However, it also significantly increases the time of vector quantization and distance estimation. For distance estimation, the methods need to decompress the codes, which introduces heavy costs. It remains to be an interesting question how to utilize the flexibility of neural networks while estimating distances efficiently. Besides quantization, hashing can also be used for vector compression. A line of studies named *signed random projection* which generate a short code for estimating the angular values between vectors via binarizing the vectors after randomization [43, 44, 47], which has its indexing phase similar to binary quantization. However, during querying, these methods map both data and query vectors into binary codes and estimate angular values via counting collisions. This disables asymmetric distance estimation and introduces errors from both data and query vectors. Furthermore, they cannot be easily adopted to cases of using more than 1 bit per dimension. Besides, there have been methods for lossless compression of floating-point arrays [40–42]. They retain perfect accuracy and would require a decompression step to recover the raw vectors for computing distances. As a comparison, quantization provides lossy compression and mostly supports distances estimation without decompression. Thus, it is likely that these methods are used for different purposes.

Based on the discussions above, incorporating learning into the quantization pipeline is a tricky question. On the one hand, enlarging the flexibility of learning usually degenerates the efficiency of vector quantization and distance estimation. On the other hand, optimizing a codebook within a larger search space does not necessarily produce better space-accuracy trade-off. Furthermore, although learning is powerful in exploring the properties of a dataset, the learning-based methods can hardly provide theoretical error bounds and are observed to fail disastrously [9]. This motivates us to explore the possibility of enhancing a method with learning while retaining rigorous theoretical error bounds. To this end, next, we will introduce RaBitQ, the first practical algorithm which achieves asymptotically optimal theoretical error bounds. Then, we will discuss the possibility of enhancing it with learning-based techniques.

3 RaBitQ: Practical and Asymptotically Optimal Quantization

RaBitQ develops a practical and asymptotically optimal algorithm of quantization with an arbitrary compression rate [9, 10]. It first normalizes the raw data vector and query vector $\mathbf{o}_r$ and $\mathbf{q}_r$ based on a centering vector $\mathbf{c}$, i.e., we take the normalized vector $\mathbf{o} := \frac{\mathbf{o}_r - \mathbf{c}}{\|\mathbf{o}_r - \mathbf{c}\|}$ and $\mathbf{q} := \frac{\mathbf{q}_r - \mathbf{c}}{\|\mathbf{q}_r - \mathbf{c}\|}$. We next reduce the original question of estimating distances between the raw vectors to the question of estimating the inner product of the unit vectors based on the following equation.

$$\|\mathbf{o}_r - \mathbf{q}_r\|^2 = \|(\mathbf{o}_r - \mathbf{c}) - (\mathbf{q}_r - \mathbf{c})\|^2 = \|\mathbf{o}_r - \mathbf{c}\|^2 + \|\mathbf{q}_r - \mathbf{c}\|^2 - 2 \cdot \|\mathbf{o}_r - \mathbf{c}\| \cdot \|\mathbf{q}_r - \mathbf{c}\| \cdot \langle \mathbf{q}, \mathbf{o} \rangle \quad (4)$$

Note that for nearest neighbor search, $\|\mathbf{o}_r - \mathbf{c}\|$ can be computed and stored before any queries come, and $\|\mathbf{q}_r - \mathbf{c}\|$ can be computed once and used for different data vectors $\mathbf{o}_r$ so that the amortized cost for each data vector is negligible. Thus, we focus on the quantization of unit vectors (i.e., $\mathbf{q}$ and $\mathbf{o}$) and the estimation of their inner product.

3.1 Insights

We first introduce a key insight behind the RaBitQ method. We note that RaBitQ achieves the asymptotically optimal performance. One of the key reasons behind the optimality is that RaBitQ fully utilizes a unique property/phenomenon in high-dimensional spaces, namely *concentration of measure* [24]. There have been vast theoretical studies on this phenomenon over the past decades [2, 25–27]. In this paper, however, instead of discussing the principle in rigorous mathematical languages, let us start from simple examples to build an intuitive understanding on the counter-intuitive phenomenon.

Consider the following scenario: we are interested in determining the value of the projection of a unit vector $\mathbf{x}$ onto a specific vector—for simplicity, let's choose the first coordinate of the vector, represented as $\mathbf{x}[0]$. However, due to certain constraints, we do not have direct access to this value. Therefore, our goal is to estimate its approximate range without performing any explicit computations. Basically, since the vector $\mathbf{x}$ is a unit vector, and with no further information, the only inference we can make is that the value of $\mathbf{x}[0]$ must fall within the interval $[-1, 1]$.

Next, let us investigate the case, where the vector $\mathbf{x}$ follows the uniform distribution on the unit sphere. To better understand the behavior of $\mathbf{x}[0]$ in this case, we generate 10^5 random vectors following this distribution. The empirical distribution of $\mathbf{x}[0]$ is plotted in Figure 2. On the left panel, we illustrate the case for a 3-dimensional vector. This scenario is intuitive: for a unit vector, $\mathbf{x}[0]$ can take any value within the range $[-1, 1]$. On the right panel, however, the situation becomes far less intuitive when the vector has 1,000 dimensions. While the theoretically possible range of $\mathbf{x}[0]$ remains $[-1, 1]$, the figure reveals a striking phenomenon: the values of $\mathbf{x}[0]$ are highly concentrated around 0. This unexpected behavior highlights a fundamental distinction between low-dimensional and high-dimensional spaces: in high-dimensional spaces, randomness (e.g., the uniform distribution of $\mathbf{x}$ on the unit sphere) can lead to surprisingly certainty (i.e., the concentration of $\mathbf{x}[0]$ around 0).

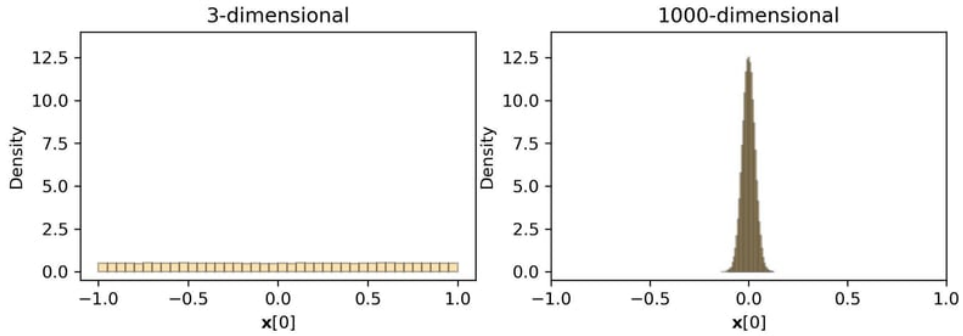


Figure 2: The Empirical Distribution of $\mathbf{x}[0]$.

This example typically demonstrates the phenomenon of concentration of measure in high-dimensional spaces. Formally, based on the seminal Johnson-Lindenstrauss Lemma [25] (JL Lemma), with probability at least 99.9%, the value $\mathbf{x}[0]$ in this example is unlikely to deviate from 0 by $\Omega(\frac{1}{\sqrt{D}})$, where D is the dimensionality. For more theoretical studies around the phenomenon of concentration of measure, we refer readers to comprehensive surveys and textbooks [24, 27].

The concentration phenomenon in high-dimensional spaces leads to intriguing implications. In

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

This insight creates opportunities to enhance algorithms by leveraging this "free information". One area, where this can be especially advantageous is quantization. By effectively harnessing this phenomenon, we can achieve significant improvements without incurring additional costs.

3.2 Codebook

3.2.1 The Codebook for 1-Bit Quantization

We first consider the case where we quantize a D-dimensional vector to a D-bit string, i.e., it is the quantization with 1-bit per dimension. Recall that we have normalized the raw vectors into unit vectors, and thus we shall focus on quantizing vectors on the unit sphere. Furthermore, as is discussed in Section 2, the construction of the codebook significantly impacts the efficiency of distance estimation. To ensure that the distance estimation can be realized with bitwise operations, instead of learning a codebook as PQ and AQ do, we take a hypercube, a special case of finite uniform grids of SQ, and align it onto the unit sphere to form the codebook, which is illustrated in Figure 3. In addition, recall that we target to utilize the "free information" which comes from randomness. Therefore, we inject the codebook some randomness by randomly rotating it, i.e., we sample a random orthogonal matrix P (a type of Johnson-Lindenstrauss Transformation) and conceptually take the rotated vectors in the hypercube as the final codebook.

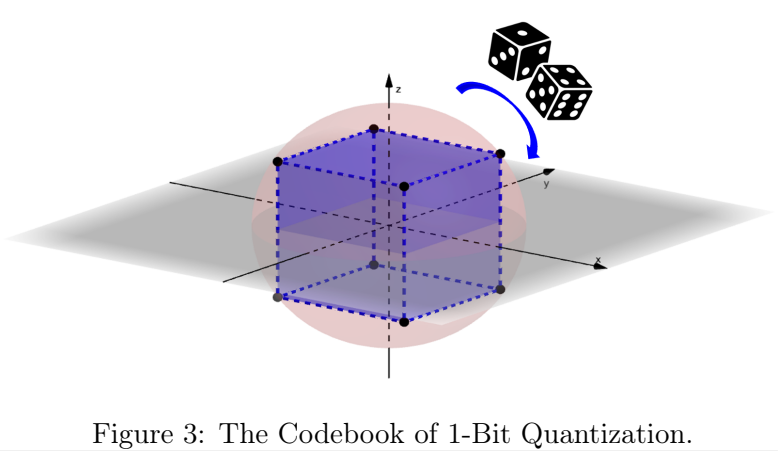


Figure 3: The Codebook of 1-Bit Quantization.

Based on this codebook, we note that the quantization process for a vector is efficient and simple. Specifically, for a given vector, we first apply an inverse rotation to it. In practice, the inverse rotation can be efficiently computed in $O(D \log D)$ time using Fast Johnson-Lindenstrauss Transformation algorithms such as Fast Walsh-Hadamard Transformation [66–68] and Kac's Walk [69]. The codeword (of the codebook before rotation) whose coordinates share the same signs as the rotated vector is the one that best approximates the original vector. By recording the sign of each coordinate as the quantization code, we effectively quantize a D-dimensional vector into a D-bit string, completing the process.

3.2.2 The Codebook for B-Bit Quantization

When extending the quantization to B bits per dimension, the construction of the codebook becomes more complex. Specifically, as is discussed in Section 2, to support efficient distance estimation based on arithmetic operations of unsigned integers, a quantization algorithm needs to adopt finite uniform grids (i.e., the codebook used in SQ) as the codebook. However, unlike 1-bit quantization, simply shifting the codebook cannot align all the vectors onto the unit sphere. To address this issue, we map the codebook onto the unit sphere through normalization, as illustrated in Figure 4. This figure provides an example of the quantization codebook when $B = 2$ in the 2-dimensional space. The empty blue points in the left panel represent a set of vectors on finite uniform grids. The solid red points in the right panel represent the normalized vectors. Applying a random rotation on the red points yields the codebook.

However, based on this codebook, the quantization process for a vector becomes challenging. For the original codebook, rounding with SQ can be easily performed for each dimension to identify the nearest vector. For the normalized codebook, the result of rounding no longer guarantees to best approximate the vector. Figure 5a provides such an example. Specifically, the purple triangle represents the vector X which we aim to quantize. In the original codebook, vector A is the result of performing rounding with SQ on X. However, in the normalized codebook, vector B is the one which best approximates X, i.e., its corresponding normalized vector is closer to X. We observe that while the optimal vector may not be found by directly rounding the vector, rescaling the vector before rounding can help locate the optimal vector. To illustrate this, consider the example in Figure 5b, where we plot a rescaled vector tX with another purple triangle. Here, the rescaling factor t is a positive number. After rescaling X and performing SQ, we successfully identified the optimal vector B in the codebook. Furthermore, we have formally proved that for any given vector, there always exists a rescaling factor such that rounding the rescaled vector guarantees the identification of the optimal vector in the codebook [10]. Therefore, to identify the optimal vector, we can explore various rescaling factors. For each rescaling factor, we determine the nearest vector by rounding with SQ and calculate the corresponding quantization error. By comparing these errors, we can ultimately determine the optimal rescaling factor and the optimal quantized vector. Based on this idea, we propose an algorithm which guarantees to find the optimal quantized vector in $O(2^B \cdot D \log D)$ time, which is good enough for practical usage as the algorithm is designed for vector compression and B is small. Additionally, when B is large, the algorithm can be further accelerated to $O(D)$ with some approximation. The detailed quantization algorithm can be found in [10].

In other words, the codebook is constructed by shifting, normalizing, and randomly rotating finite uniform grids, which are the codebooks used in SQ. It is worth noting that the quantization algorithm can also be equivalently viewed as a method for exploring and optimizing the parameters of SQ. In

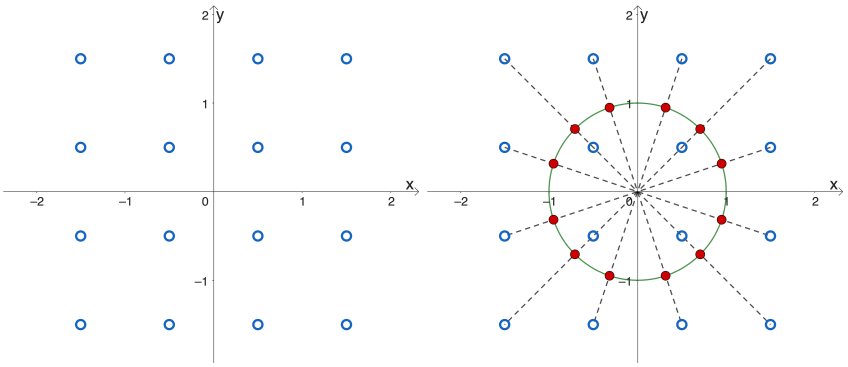


Figure 4: The Codebook of B-Bit Quantization.

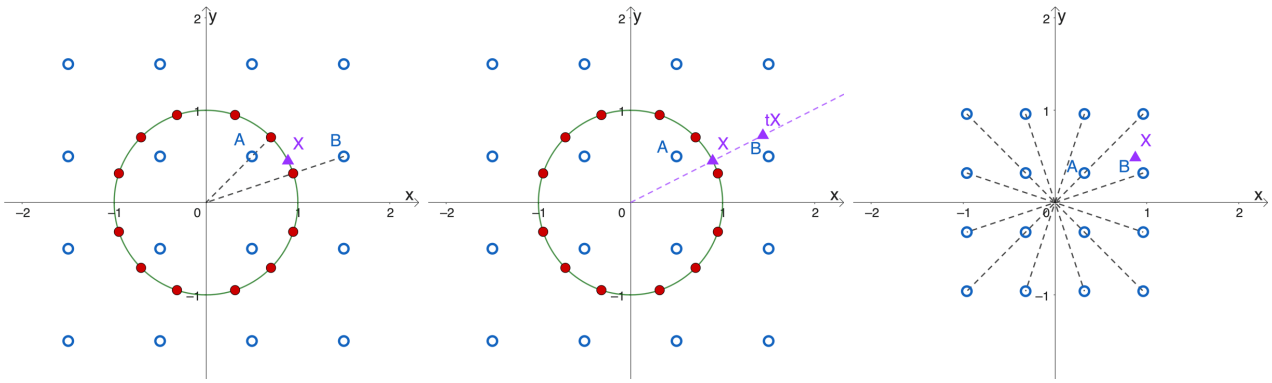


Figure 5: An Example of the Quantization Process for B-Bit Codebooks.

particular, rescaling the input vectors can be equivalently viewed as rescaling the codebook itself. In Figure 5a, rounding X onto the codebook yields the vector A. In Figure 5c, we keep the vector X unchanged and rescale the codebook. In this scenario, rounding X onto the rescaled codebook yields the rescaled vector B. The distance between X and B in Figure 5c is much smaller than that between X and A in Figure 5a. Therefore, our algorithm can be equivalently explained as performing SQ by testing different parameters on a per-vector basis, computing the quantization error for each, and selecting the optimal parameter and codeword to minimize the error. Mathematically, rescaling a vector by a factor of t is equivalent to rescaling a codebook by $1/t$. Additionally, the codebook for 1-bit quantization can be seen as a special case of the codebook for B-bit quantization with $B = 1$.

3.3 Distance Estimator

We have now completed the construction of the codebook and the quantization of the data vectors. Recall that $\mathbf{o}$ represents the data vector to be quantized, $\mathbf{q}$ is a query vector and P is a sample from a random orthogonal matrix. The quantized data vector is denoted as $\bar{\mathbf{o}}$, and its representation using unsigned integers is given by $\mathbf{x}_u$. Formally, we have $\bar{\mathbf{o}} = P(\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D)/\|\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D\|$ [9, 10]. Building on the quantized data vector, we now introduce an estimator for the inner product $\langle \mathbf{o}, \mathbf{q} \rangle$.

3.3.1 The Construction of the Estimator

To achieve this, we first analyze the geometric relationship among the vectors $\mathbf{o}$, $\bar{\mathbf{o}}$ and $\mathbf{q}$. In particular, we observe that although $\mathbf{o}$, $\bar{\mathbf{o}}$ and $\mathbf{q}$ are vectors in high-dimensional spaces, estimating $\langle \mathbf{o}, \mathbf{q} \rangle$ only requires focusing on the 2-dimensional subspace that contains $\mathbf{o}$ and $\mathbf{q}$, as illustrated in Figure 6. Here, $\mathbf{e}_1$ is a unit vector that is orthogonal to $\mathbf{o}$ and lies within the subspace. By analyzing the geometric relationships among the vectors in the subspace, we derive the following equation for the inner product of the vectors.

$$\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{q}, \mathbf{o} \rangle + \langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle \cdot \langle \mathbf{q}, \mathbf{e}_1 \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{q}, \mathbf{o} \rangle + \langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle \cdot \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2} \tag{5}$$

By transforming the equation, we have the following equation.

$$\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} = \langle \mathbf{o}, \mathbf{q} \rangle + \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2} \cdot \frac{\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} \tag{6}$$

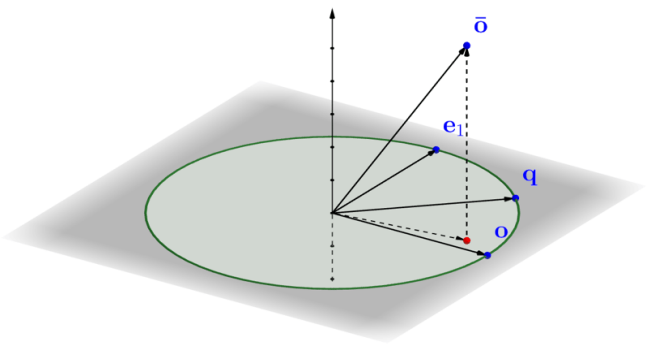


Figure 6: Geometric Relationship among the Vectors.

Note that $\langle \mathbf{o}, \mathbf{q} \rangle$ is our target. We treat the first term $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ as an estimator, and thus the last term represents the error. Recall that during codebook construction, we have sampled a random orthogonal matrix P, applied it to all codewords and generated a *randomized* codebook. The vector $\bar{\mathbf{o}}$ is a vector selected from this randomized codebook, making it a random vector. The inner products $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ represent the projections of the random vector $\bar{\mathbf{o}}$ onto $\mathbf{o}$ and $\mathbf{e}_1$, respectively. As a result, these inner products are random variables. As is discussed in Section 3.1, in high-dimensional spaces, randomness can lead to the concentration phenomenon. We rigorously analyze the distribution, in particular, the extent of concentration of $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$. Based on the analysis, we have proven that the error term has 0 expectation, indicating that the estimator is unbiased [9].

$$\mathbb{E}\left[\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}\right] = \langle \mathbf{o}, \mathbf{q} \rangle \tag{7}$$

Additionally, we have proven that the space-accuracy trade-off of our algorithm achieves the asymptotical optimality, which was established by a prior theoretical study [2]. This makes our algorithm the first to achieve the optimality as a practically applicable algorithm, to the best of our knowledge. The specific theoretical result is presented as follows [10].

Theorem 3.1: Let D be the dimensionality. For $\epsilon > 0$ where $\frac{1}{\epsilon^2} \log \frac{1}{\delta} > D$, to ensure that the error of the estimator is bounded by ϵ with probability at least $1 - \delta$, it is sufficient to set $B = \Theta\left(\log\left(\frac{1}{D} \cdot \frac{1}{\epsilon^2} \log \frac{1}{\delta}\right)\right)$.

It is worth noting that the number of bits B is *logarithmic* with respect to ϵ^{-2} and is *negatively* related to the dimensionality D. This indicates a counter-intuitive fact: the higher the dimensionality is, under the same compression rates, the smaller the error would be.

3.3.2 The Computation of the Estimator

The question of estimating distances has been reduced to the one of computing $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ / $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$, (i.e., the estimator). Note that $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ is independent of the query. It can be precomputed before querying. Recall that $\bar{\mathbf{o}}$ is a vector in the codebook which is generated by shifting, normalizing and rotating finite unit grids, i.e., the vectors of B-bit unsigned integer. We denote the D-dimensional vector whose all coordinates are equal to 1 by $\mathbf{1}_D$. Additionally, let $\mathbf{q}' := P^{-1}\mathbf{q}$. To support the computation of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ with arithmetic operations of unsigned integers, we transform $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ as follows.

$$\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \left\langle P\frac{\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D}{\|\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D\|}, \mathbf{q} \right\rangle = \frac{1}{\|\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D\|}\left(\langle \mathbf{x}_u, \mathbf{q}' \rangle - \frac{2^B - 1}{2} \sum_{i=1}^{D} \mathbf{q}'[i]\right) \tag{8}$$

Here $\|\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D\|$ is independent of queries and can be precomputed. The variable $\sum_{i=1}^{D} \mathbf{q}'[i]$ can be computed once and shared by all data vectors. We perform quantization on $\mathbf{q}'$, i.e., we approximate it with $\Delta \cdot \mathbf{q}_u + v_l \cdot \mathbf{1}_D$. Then the computation is finally reduced to the inner product between vectors of unsigned integers as follows.

$$\left\langle \mathbf{x}_u, \mathbf{q}' \right\rangle = \langle \mathbf{x}_u, \Delta \cdot \mathbf{q}_u + v_l \cdot \mathbf{1}_D \rangle = \Delta \cdot \langle \mathbf{x}_u, \mathbf{q}_u \rangle + v_l \cdot \sum_{i=1}^{D} \mathbf{x}_u[i] \tag{9}$$

Note that number of bits used for quantizing the query $\mathbf{q}'$ should be larger than B by more than 3 bits so that the error introduced from the query side is negligible. Particularly, when $B = 1$, the computation can be realized with highly efficient bitwise operations. Note that the computation is exactly the same as SQ. Thus, RaBitQ can replace SQ seamlessly while providing better accuracy. We refer readers to the RaBitQ papers for more details [9, 10].

3.4 Summary

In summary, our RaBitQ methods provide optimized methods for binary and scalar quantization. Its distance estimation can be realized in the same way of binary and scalar quantization. Under the same compression rates, it consistently produces better accuracy than LVQ [1], the state-of-the-art variant of SQ. Thus, replacing the SQ methods with RaBitQ would bring consistent improvement. In particular, we would like to highlight that when $B = 1$ and $B = 2$, the error of RaBitQ is smaller than that of LVQ by orders of magnitude, indicating RaBitQ's significant improvement [10]. Additionally, RaBitQ primarily focuses on the compression rates ranging from 3x to 32x, with only limited discussion on settings with over 32x compression. A concurrent study of the extended RaBitQ paper, MRQ [28], focuses on achieving over 32x compression and enhancing performance using data-aware technologies. We refer readers to the MRQ paper for more detailed discussions [28]. Finally, to the best of our knowledge, RaBitQ is the first practical algorithm which achieves the asymptotic optimality established by a prior theoretical study [2].

4 Future Directions: towards Data-Aware Methods with Theoretical Error Bounds

The RaBitQ method provides practical and asymptotically optimal quantization. This result relies on the codebook construction process, where the codebook is generated based on the codebook of SQ. This codebook allows distance estimation to be efficiently computed using bitwise operations (for 1-bit quantization) and arithmetic operations on unsigned integers (for B-bit quantization). The theoretical error bound requires that the codebook is randomly rotated. Therefore, incorporating learning into either codebook construction or rotation will cause the loss of the theoretical error bounds. However, we observe that the theoretical bounds remain valid regardless of the centering vector chosen for normalization. Consequently, the normalization step can be improved without sacrificing any theoretical guarantees. This opens up the opportunity to enhance RaBitQ by incorporating advanced learning techniques into the normalization step. Nevertheless, based on the current scheme, there are still some constraints on the selection of centering vectors for normalization. Specifically, it uses the centroids of KMeans clustering [9, 10] for normalization. In this scheme, the algorithm needs to pre-process (i.e., quantize) the query vector for every centroid. This cost can be amortized if a centroid is shared by many data vectors. However, when a centroid is shared by a few data vectors only, the cost of pre-processing the query vector would be significant. This issue can be resolved with a simple trick, which is presented as

follows.

$$\|\mathbf{o_r} - \mathbf{q}_r\|^2 = \|\mathbf{o}_r - \mathbf{c}\|^2 + \|\mathbf{q}_r - \mathbf{c}\|^2 - 2\langle \mathbf{o}_r - \mathbf{c}, \mathbf{q}_r - \mathbf{c}\rangle \tag{10}$$

$$\approx \|\mathbf{o}_r - \mathbf{c}\|^2 + \|\mathbf{q}_r - \mathbf{c}\|^2 - 2\|\mathbf{o}_r - \mathbf{c}\| \cdot \|\mathbf{q}_r - \mathbf{c}\| \cdot \frac{\langle \bar{\mathbf{o}}, \mathbf{q}\rangle}{\langle \bar{\mathbf{o}}, \mathbf{o}\rangle} \tag{11}$$

$$= \|\mathbf{o}_r - \mathbf{c}\|^2 + \|\mathbf{q}_r - \mathbf{c}\|^2 - 2\|\mathbf{o}_r - \mathbf{c}\| \cdot \frac{\langle \bar{\mathbf{o}}, \mathbf{q}_r - \mathbf{c}\rangle}{\langle \bar{\mathbf{o}}, \mathbf{o}\rangle} \tag{12}$$

$$= \|\mathbf{o}_r - \mathbf{c}\|^2 + 2\|\mathbf{o}_r - \mathbf{c}\| \frac{\langle \bar{\mathbf{o}}, \mathbf{c}\rangle}{\langle \bar{\mathbf{o}}, \mathbf{o}\rangle} + \|\mathbf{q}_r - \mathbf{c}\|^2 - 2\|\mathbf{o}_r - \mathbf{c}\| \cdot \frac{\langle \bar{\mathbf{o}}, \mathbf{q}_r\rangle}{\langle \bar{\mathbf{o}}, \mathbf{o}\rangle} \tag{13}$$

The first two terms are independent of queries and can be precomputed. Thus, the only question is to compute $\langle \bar{\mathbf{o}}, \mathbf{q}_r\rangle$, where $\mathbf{q}_r$ is independent of the centroid. Thus, when a query comes, it can be pre-processed once and shared by all data vectors, even though their normalization is based on different centering vectors.

With this trick, one possible research direction is to combine RaBitQ with other types of quantization methods. In particular, it is possible to use the quantized vector of other quantization methods (e.g., additive quantization) for normalization. Specifically, for a dataset, we can first adopt additive quantization to find a quantized vector for every vector. Then, we use the quantized vector as the centering vector for normalization. Recall that additive quantization could perform competitively when using a small number of bits, due to its better flexibility of codebook construction. It may capture the data distribution with a few bits, which can enhance the normalization step with moderate costs. On the other hand, RaBitQ provides better scalability, efficiency and rigorous theoretical error bounds. Combining both methods is expected to enhance data-awareness while preserving error bounds. However, several details remain to be addressed in this direction, which should be carefully studied.

5 Conclusion

Vector quantization is fundamental for reducing the memory costs of high-dimensional vector data management. In this paper, we presented the general framework of vector quantization, which involves two critical components, the codebook and the distance estimator. We reviewed existing popular quantization schemes, and discussed how incorporating learning within their pipelines affects their costs of vector quantization and distance estimation. We provided an intuitive overview of the RaBitQ method, which serves as an optimized approach to binary and scalar quantization, and achieve asymptotically optimal theoretical error bounds. Finally, we discussed opportunities to further enhance the RaBitQ methods through learning-based normalization techniques, aiming to develop a data-aware approach with rigorous theoretical error bounds.

References

[1] Aguerrebere, Cecilia and Bhati, Ishwar Singh and Hildebrand, Mark and Tepper, Mariano and Willke, Theodore. Similarity Search in the Blink of an Eye with Compressed Indices. *VLDB*, 2023.

[2] Alon, Noga, and Klartag, Bo'az. Optimal Compression of Approximate Inner Products and Dimension Reduction. *FOCS*, 2017.

[3] André, Fabien, Kermarrec, Anne-Marie, and Le Scouarnec, Nicolas. Cache Locality is Not Enough: High-Performance Nearest Neighbor Search with Product Quantization Fast Scan. *VLDB*, 2015.

[4] Ilias Azizi, Karima Echihabi, and Themis Palpanas. ELPIS: Graph-Based Similarity Search for Scalable Data Science. In *PVLDB*, 2023.

[5] André, Fabien, Kermarrec, Anne-Marie, and Le Scouarnec, Nicolas. Accelerated Nearest Neighbor Search with Quick ADC. *ICMR '17*, 2017.

[6] André, Fabien, Kermarrec, Anne-Marie, and Le Scouarnec, Nicolas. Quicker ADC: Unlocking the Hidden Potential of Product Quantization With SIMD. *IEEE TPAMI*, 2021.

[7] Babenko, Artem, and Lempitsky, Victor. Additive Quantization for Extreme Vector Compression. *CVPR*, 2014.

[8] Gao, Jianyang, and Long, Cheng. High-Dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations. In *SIGMOD*, 2023.

[9] Gao, Jianyang, and Long, Cheng. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *SIGMOD*, 2024.

[10] Gao, Jianyang, Gou, Yutong, Xu, Yuexuan, Yang, Yongyi, Long, Cheng, and Wong, Raymond Chi-Wing. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. *CoRR*, 2024 (to appear in SIGMOD 2025).

[11] Liang, Tailin, Glossner, John, Wang, Lei, Shi, Shaobo, and Zhang, Xiaotong. Pruning and Quantization for Deep Neural Network Acceleration: A Survey. *Neurocomputing*, 2021.

[12] Gholami, Amir, Kim, Sehoon, Dong, Zhen, Yao, Zhewei, Mahoney, Michael W., and Keutzer, Kurt. A Survey of Quantization Methods for Efficient Neural Network Inference. *Low-Power Computer Vision*, 2022.

[13] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. In *PVLDB*, 2023.

[14] Jiuqi Wei, Xiaodong Lee, Zhenyu Liao, Themis Palpanas, and Botao Peng. Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. In *SIGMOD*, 2025.

[15] Wang, Jianguo and Yi, Xiaomeng and Guo, Rentong and Jin, Hai and Xu, Peng and Li, Shengjun and Wang, Xiangyu and Guo, Xiangzhou and Li, Chengming and Xu, Xiaohai and Yu, Kun and Yuan, Yuxing and Zou, Yinghao and Long, Jiquan and Cai, Yudong and Li, Zhenxiang and Zhang, Zhifeng and Mo, Yihua and Gu, Jun and Jiang, Ruiyi and Wei, Yi and Xie, Charles. Milvus: A Purpose-Built Vector Data Management System. *SIGMOD*, 2021.

[16] Martinez, Julieta, Zakhmi, Shobhit, Hoos, Holger H., and Little, James J. LSQ++: Lower Running Time and Higher Recall in Multi-Codebook Quantization. *ECCV*, 2018.

[17] Jegou, Herve, Douze, Matthijs, and Schmid, Cordelia. Product quantization for nearest neighbor search. *IEEE TPAMI*, 2010.

[18] Ge, Tiezheng, He, Kaiming, Ke, Qifa, and Sun, Jian. Optimized product quantization for approximate nearest neighbor search. *CVPR*, 2013.

[19] Wang, Jingdong, Zhang, Ting, Song, Jingkuan, Sebe, Nicu, and Shen, Heng Tao. A Survey on Learning to Hash. *IEEE TPAMI*, 2018.

[20] Chen, Cheng, Jin, Chenzhe, Zhang, Yunan, Podolsky, Sasha, Wu, Chun, Wang, Szu-Po, Hanson, Eric, Sun, Zhou, Walzer, Robert, and Wang, Jianguo. SingleStore-V: An Integrated Vector Database System in SingleStore. *VLDB*, 2024.

[21] Douze, Matthijs, Guzhva, Alexandr, Deng, Chengqi, Johnson, Jeff, Szilvasy, Gergely, Mazaré, Pierre-Emmanuel, Lomeli, Maria, Hosseini, Lucas, and Jégou, Hervé. The Faiss library. *CoRR*, 2024.

[22] Qitong Wang, Ioana Ileana, and Themis Palpanas. LeaFi: Data Series Indexes on Steroids with Learned Filters. In *SIGMOD*, 2025.

[23] Martinez, Julieta, Clement, Joris, Hoos, Holger H., and Little, James J. Revisiting Additive Quantization. *ECCV*, 2016.

[24] Vershynin, Roman. High-Dimensional Probability: An Introduction with Applications in Data Science. *Cambridge University Press*, 2018.

[25] Johnson, William B., and Lindenstrauss, Joram. Extensions of Lipschitz mappings into a Hilbert space 26. *Contemporary Mathematics*, 1984.

[26] Larsen, Kasper Green, and Nelson, Jelani. Optimality of the Johnson-Lindenstrauss lemma. *FOCS*, 2017.

[27] Freksen, Casper Benjamin. An Introduction to Johnson-Lindenstrauss Transforms. *CoRR*, 2021.

[28] Yang, Mingyu, Li, Wentao, and Wang, Wei. Fast High-dimensional Approximate Nearest Neighbor Search with Efficient Index Time and Space. *CoRR*, 2024.

[29] Schäfer, Patrick, Brand, Jakob, Leser, Ulf, Peng, Botao, and Palpanas, Themis. Fast and Exact Similarity Search in less than a Blink of an Eye. *CoRR*, 2024.

[30] Yang, Ming, Cai, Yuzheng, and Zheng, Weiguo. CSPG: Crossing Sparse Proximity Graphs for Approximate Nearest Neighbor Search. *NeurIPS*, 2024.

[31] Deng, Liwei, Chen, Penghao, Zeng, Ximu, Wang, Tianfu, Zhao, Yan, and Zheng, Kai. Efficient Data-aware Distance Comparison Operations for High-Dimensional Approximate Nearest Neighbor Search. *CoRR*, 2024.

[32] Zheng, Bolong, Zhao, Xi, Weng, Lianggui, Hung, Nguyen Quoc Viet, Liu, Hang, and Jensen, Christian S. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *VLDB*, 2020.

[33] Jiang, Wenqi, Li, Shigang, Zhu, Yu, De Fine Licht, Johannes, He, Zhenhao, Shi, Runbin, Renggli, Cedric, Zhang, Shuai, Rekatsinas, Theodoros, Hoefler, Torsten, and Alonso, Gustavo. Co-design Hardware and Algorithm for Vector Search. *SC*, 2023.

[34] Chen, Meng, Zhang, Kai, He, Zhenying, Jing, Yinan, and Wang, X. Sean. RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. *VLDB*, 2024.

[35] Wang, Mengzhao, Xu, Weizhi, Yi, Xiaomeng, Wu, Songlin, Peng, Zhangyang, Ke, Xiangyu, Gao, Yunjun, Xu, Xiaoliang, Guo, Rentong, and Xie, Charles. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *SIGMOD*, 2024.

[36] Wang, Zeyu, Wang, Qitong, Cheng, Xiaoxing, Wang, Peng, Palpanas, Themis, and Wang, Wei. Steiner-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. *PVLDB*, 2024.

[37] Chen, Yongjian, Guan, Tao, and Wang, Cheng. Approximate Nearest Neighbor Search by Residual Vector Quantization. *Sensors*, 2010.

[38] Morozov, Stanislav, and Babenko, Artem. Unsupervised Neural Quantization for Compressed-Domain Similarity Search. *ICCV*, 2019.

[39] Huijben, Iris A. M., Douze, Matthijs, Muckley, Matthew J., van Sloun, Ruud, and Verbeek, Jakob. Residual Quantization with Implicit Neural Codebooks. *ICML*, 2024.

[40] Afroozeh, Azim, Kuffo, Leonardo X., and Boncz, Peter. ALP: Adaptive Lossless Floating-Point Compression. *SIGMOD*, 2023.

[41] Pelkonen, Tuomas, Franklin, Scott, Teller, Justin, Cavallaro, Paul, Huang, Qi, Meza, Justin, and Veeraraghavan, Kaushik. Gorilla: a fast, scalable, in-memory time series database. *VLDB*, 2015.

[42] Liakos, Panagiotis, Papakonstantinopoulou, Katia, and Kotidis, Yannis. Chimp: efficient lossless floating point compression for time series databases. *VLDB*, 2022.

[43] Charikar, Moses S. Similarity Estimation Techniques from Rounding Algorithms. *STOC*, 2002.

[44] Dubey, Punit Pankaj, Verma, Bhisham Dev, Pratap, Rameshwar, and Kang, Keegan. Improving sign-random-projection via count sketch. *UAI*, 2022.

[45] Yang, Wen, Li, Tao, Fang, Gai, and Wei, Hong. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension. *SIGMOD*, 2020.

[46] Mohoney, Jason, Pacaci, Anil, Chowdhury, Shihabur Rahman, Mousavi, Ali, Ilyas, Ihab F., Minhas, Umar Farooq, Pound, Jeffrey, and Rekatsinas, Theodoros. High-Throughput Vector Similarity Search in Knowledge Graphs. *SIGMOD*, 2023.

[47] Ji, Jianqiu, Li, Jianmin, Yan, Shuicheng, Zhang, Bo, and Tian, Qi. Super-Bit Locality-Sensitive Hashing. *NeurIPS*, 2012.

[48] Liu, Ying, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. A survey of content-based image retrieval with high-level semantics. *Pattern recognition* 40, no. 1 (2007): 262-282.

[49] Joshua Engels, Benjamin Landrum, Shangdi Yu, Laxman Dhulipala, and Julian Shun. Approximate Nearest Neighbor Search with Window Filters. *arXiv preprint* arXiv:2402.00943 (2024).

[50] Schafer, J. Ben, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*, pp. 291-324. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[51] Asai, Akari, Sewon Min, Zexuan Zhong, and Danqi Chen. Tutorial Proposal: Retrieval-based Language Models and Applications. In *The 61st Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, p. 41. 2023.

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

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

[54] Malkov, Yu A., and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, no. 4 (2018): 824-836.

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

[56] Guo, Ruiqi, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pp. 3887-3896. PMLR, 2020.

[57] Weber, Roger, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, vol. 98, pp. 194-205. 1998.

[58] Paparrizos, John, Edian, Ikraduya, Liu, Chunwei, Elmore, Aaron J., and Franklin, Michael J. Fast Adaptive Similarity Search through Variance-Aware Quantization. *ICDE*, 2022.

[59] Guo, Ruiqi, Sun, Philip, Lindgren, Erik, Geng, Quan, Simcha, David, Chern, Felix, and Kumar, Sanjiv. Accelerating large-scale inference with anisotropic vector quantization. *ICML*, 2020.

[60] Ferhatosmanoglu, Hakan, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, pp. 202-209. 2000.

[61] Wei, Jiuqi, Peng, Botao, Lee, Xiaodong, and Palpanas, Themis. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *VLDB*, 2024.

[62] Lu, Kejing, Kudo, Mineichi, Xiao, Chuan, and Ishikawa, Yoshiharu. HVS: hierarchical graph structure based on Voronoi diagrams for solving approximate nearest neighbor search. *VLDB*, 2021.

[63] Song, Yitong, Wang, Kai, Yao, Bin, Chen, Zhida, Xie, Jiong, and Li, Feifei. Efficient Reverse k Approximate Nearest Neighbor Search Over High-Dimensional Vectors. *ICDE*, 2024.

[64] Zuo, Chaoji, Qiao, Miao, Zhou, Wenchao, Li, Feifei, and Deng, Dong. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *SIGMOD*, 2024.

[65] Su, Yongye, Sun, Yinqi, Zhang, Minjia, and Wang, Jianguo. Vexless: A Serverless Vector Data Management System Using Cloud Functions. *SIGMOD*, 2024.

[66] Andoni, Alexandr, Indyk, Piotr, Laarhoven, Thijs, Razenshteyn, Ilya, and Schmidt, Ludwig. Practical and Optimal LSH for Angular Distance. *NeurIPS*, 2015.

[67] FALCONN-LIB. Library for Fast Fast Hadamard Transform. 2015. Available at: `https://github.com/FALCONN-LIB/FFHT`. Accessed: 17 Apr, 2024.

[68] Ailon, Nir, and Chazelle, Bernard. The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors. *SIAM Journal on Computing*, 2009.

[69] Jain, Vishesh, Pillai, Natesh S., Sah, Ashwin, Sawhney, Mehtaab, and Smith, Aaron. Fast and Memory-Optimal Dimension Reduction Using Kac's Walk. *The Annals of Applied Probability*, 2022.