SAQ: Pushing the Limits of Vector Quantization through Code Adjustment and Dimension Segmentation

HUI LI, The Chinese University of Hong Kong, Hong Kong SAR
SHIYUAN DENG, Huawei Cloud, China
XIAO YAN*, Institute for Math & AI, Wuhan, Wuhan University, China
XIANGYU ZHI, The Chinese University of Hong Kong, Hong Kong SAR
JAMES CHENG, The Chinese University of Hong Kong, Hong Kong SAR

Approximate Nearest Neighbor Search (ANNS) plays a critical role in applications such as search engines, recommender systems, and RAG for LLMs. Vector quantization (VQ), a crucial technique for ANNS, is commonly used to reduce space overhead and accelerate distance computations. However, despite significant research advances, state-of-the-art VQ methods still face challenges in balancing encoding efficiency and quantization accuracy. To address these limitations, we propose a novel VQ method called SAQ. To improve accuracy, SAQ employs a new dimension segmentation technique to strategically partition PCA-projected vectors into segments along their dimensions. By prioritizing leading dimension segments with larger magnitudes, SAQ allocates more bits to high-impact segments, optimizing the use of the available space quota. An efficient dynamic programming algorithm is developed to optimize dimension segmentation and bit allocation, ensuring minimal quantization error. To speed up vector encoding, SAQ devises a code adjustment technique to first quantize each dimension independently and then progressively refine quantized vectors using a coordinate-descent-like approach to avoid exhaustive enumeration. Extensive experiments demonstrate SAQ's superiority over classical methods (e.g., PQ, PCA) and recent state-of-the-art approaches (e.g., LVQ, Extended RabitQ). SAQ achieves up to 80% reduction in quantization error and accelerates encoding speed by over 80× compared to Extended RabitQ.

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

Additional Key Words and Phrases: Vector quantization, nearest neighbor search, vector database

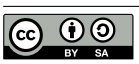
ACM Reference Format:
Hui Li, Shiyuan Deng, Xiao Yan, Xiangyu Zhi, and James Cheng. 2025. SAQ: Pushing the Limits of Vector Quantization through Code Adjustment and Dimension Segmentation. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 359 (December 2025), 25 pages. https://doi.org/10.1145/3769824

1 Introduction

With the proliferation of machine learning, embedding models [10, 29, 41, 42] are widely used to map diverse data objects, including images, videos, texts, e-commerce goods, genes, and proteins, to high-dimensional vector embeddings that encode their semantic information. A core operation

*Dr. Xiao Yan is the corresponding author.

Authors' Contact Information: Hui Li, The Chinese University of Hong Kong, Hong Kong SAR, hli@cse.cuhk.edu.hk; Shiyuan Deng, Huawei Cloud, China, dengshiyuan@huawei.com; Xiao Yan, Institute for Math & AI, Wuhan, Wuhan University, China, yanxiaosunny@whu.edu.cn; Xiangyu Zhi, The Chinese University of Hong Kong, Hong Kong SAR, xyzhi24@cse.cuhk.edu.hk; James Cheng, The Chinese University of Hong Kong, Hong Kong SAR, jcheng@cse.cuhk.edu.hk.



ACM 2836-6573/2025/12-ART359
https://doi.org/10.1145/3769824

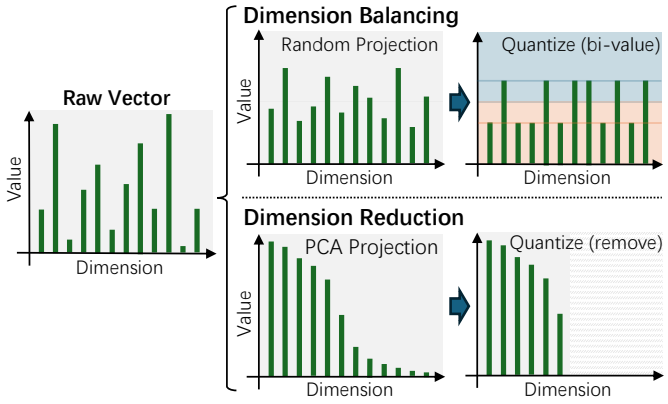


Fig. 1. Illustration of dimension balancing and dimension reduction. Bar height is the magnitude of vector dimension.

on these vectors is *nearest neighbor search* (*NNS*) [11, 36, 39, 40, 43, 49, 55], which retrieves vectors that are the most similar to a query vector from a vector dataset. NNS underpins many important applications including content search (e.g., for images and videos), recommender systems, bioinformatics, and retrieval-argumented generation (RAG) for LLMs [26]. However, exact NNS requires a linear scan in high-dimensional space, rendering it impractical for large-scale datasets. To address this, *approximate NNS* (*ANNS*) has been widely adopted [21, 53], which trades exactness for efficiency by returning most of the top-k nearest neighbors. Many vector indexes [31, 56] and vector databases [11, 39, 40, 43, 49] support ANNS as the core functionality.

Vector quantization (or vector compression) maps each vector $\mathbf{o}$ to a smaller approximate vector $\bar{\mathbf{o}}$ and is a key technique for efficient ANNS. With billions of vectors and each vector having thousands of dimensions, vector datasets can be large (e.g., in TBs) and vector quantization helps reduce space consumption. Moreover, vector quantization enables us to compute approximate distance (i.e., $\|\mathbf{q}-\bar{\mathbf{o}}\|$, where $\mathbf{q}$ is the query vector) faster than the exact distance (i.e., $\|\mathbf{q}-\mathbf{o}\|$). Since distance computation dominates the running time of vector indexes [17], vector quantization effectively accelerates ANNS. The goal of vector quantization is to minimize the *relative error* (defined as $\left|\|\mathbf{q}-\bar{\mathbf{o}}\|^2-\|\mathbf{q}-\mathbf{o}\|^2\right|/\|\mathbf{q}-\mathbf{o}\|^2$) under a given space quota (i.e., measured by compression ratio w.r.t. the original vector or the average number of bits used for each dimension) such that approximate distance matches exact distance.

Many vector quantization methods have been proposed. PQ [5, 23, 34, 51], OPQ [19], and LOPQ [25] partition the D-dimensional vector space into subspaces and use K-means to learn vector codebooks for each subspace. AQ [5], RQ [7], LSQ [33, 34], and TreeQ [6, 13] further improve the structures and learning methods of the vector codebooks. Some other methods learn query-dependent codebooks to better approximate the query-vector distance [20, 30]. The state-of-the-art vector quantization methods adopt two types of designs as illustrated in Figure 1. The first one is *dimension balancing*, which uses a random orthonormal matrix R to project a vector x and then quantizes each dimension of Rx to an integer. The representative is RaBitQ [16, 18], which quantizes each projected vector dimension with 1 bit and provides unbiased distance estimation. The second type is *dimension reduction*, which uses a PCA projection matrix P to project a vector x such that the leading dimensions correspond to large eigenvalues and thus have large magnitudes. Then, the small tailing dimensions are discarded.

Despite the progresses, we observe two limitations in the state-of-the-art vector quantization methods.

- **High quantization complexity**: Currently, RaBitQ [18] and extended RaBitQ (E-RaBitQ) [16] achieve the best accuracy. However, RaBitQ can only use 1 bit for each vector dimension, which limits accuracy, while the quantization complexity of E-RaBitQ is $O(2^B \cdot D \log D)$ where a vector has D dimensions and each dimension uses B bits. This is because E-RaBitQ does not handle each vector dimension independently, but an enumeration is required to decide the optimal quantized vector to approximate the original vector. Our profiling shows that when $B = 9$ and $D = 3,072$, quantizing 1 billion vectors with E-RaBitQ takes more than 3,600 CPU hours.
- **Contradictory designs**: Dimension balancing attempts to make vector dimensions similar in magnitude such that the same number of bits can be used for each dimension. In contrast, dimension reduction makes the vector dimensions skewed in magnitude so that a vector can be approximated by its leading dimensions. These two designs are fundamentally contradictory, posing problems about selecting the better method for different applications or considering synergistic potential for enhanced performance.

The inherent conflict between the two dominant design paradigms in existing quantization methods makes it difficult to further improve the efficiency of quantization while retaining the accuracy of vector search. To resolve this challenge, we propose a novel vector quantization method, *SAQ*, that overcomes the limitations of state-of-the-art methods through two key innovations: *code adjustment* and *dimension segmentation*. One striking contribution of SAQ is: SAQ significantly advances *both* quantization efficiency and vector search accuracy of currently best methods. As shown in Figure 2, SAQ achieves significantly lower quantization errors than RaBitQ[1] with the same space quota, while delivering dramatic speed-ups of over 80x faster quantization.

SAQ introduces a *code adjustment* technique to accelerate vector quantization. In particular, after random orthonormal projection, SAQ first quantizes each dimension of a vector independently using B bits. Observing that quantization accuracy depends on how well the original vector x and quantized vector $\tilde{x}$ align in their directions (measured by cosine similarity), we propose to adjust the quantized code for each dimension of $\tilde{x}$ to better align with x. This yields a complexity of $O(D)$ to quantize each vector, avoiding RaBitQ's code enumeration which requires $O(2^B \cdot D \log D)$. We call the new quantization method *CAQ* and show that CAQ achieves the same quantization accuracy as RaBitQ, as code adjustment essentially conducts coordinate descent [52] to optimize $\tilde{x}$.

SAQ utilizes a *dimension segmentation* technique to bridge dimension reduction and dimension balancing for improved accuracy. After PCA projection, the vector dimensions are partitioned into segments, and CAQ is applied independently to each dimension segment. The key is to allocate more bits to the segments of leading (high-magnitude) dimensions to improve accuracy, while segments of trailing dimensions use fewer bits or are discarded. We develop a dynamic programming algorithm to optimally allocate bits across dimension segments to minimize quantization error under a total space quota $B \cdot D$. We show that dimension segmentation ensures that SAQ's approximation error cannot be larger than RaBitQ's.

We further enhance SAQ with a set of novel optimizations. First, we design a distance estimators based on the quantized vectors and develop single-instruction-multiple-data (SIMD) implementations for efficient distance computation. Second, CAQ supports *progressive distance approximation*, enabling us to take the first $b < B$ bits for each dimension while still yielding a valid quantized vector (with reduced accuracy). Third, SAQ supports multi-stage distance estimation, which gives lower and upper bounds for distances using the segmented approximate vectors. The two properties are useful for pruning in ANNS [16, 17].

[1]In the subsequent discussion, we also refer to E-RaBitQ as RabitQ for simplicity as most of our discussions are related to E-RaBitQ.

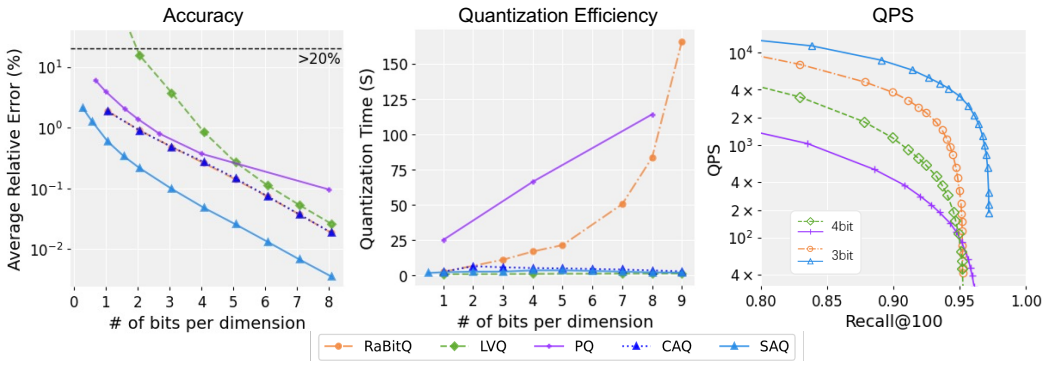


Fig. 2. The performance of SAQ and representative baselines for the GIST dataset. Note that RaBitQ refers to extended RaBitQ (same for other experiments).

We conduct extensive experiments to evaluate SAQ and compare it with four representative vector quantization methods. The results show that under the same space quota, SAQ consistently achieves lower approximation error than the baselines. In particular, to achieve the same accuracy as 8-bit RaBitQ, SAQ only needs 5-6 bits for each dimension, which is also shown in Figure 2. When both RaBitQ and SAQ use 8 bits for each dimension, the approximation errors of SAQ are usually below 50% of E-RaBitQ. Moreover, SAQ can accelerate the vector quantization process of RaBitQ by up to 80x. We also show that SAQ's accuracy gains translate to higher query throughput for ANNS, and that our two key designs, i.e., code adjustment and dimension segmentation, are effective.

Our key contributions are as follows.

- We identify the critical limitations in state-of-the-art vector quantization methods, i.e., prolonged quantization time and contradictory design paradigms.
- We propose SAQ, which pushes the limits of current vector quantization methods in both accuracy and efficiency. SAQ comes with two key designs: code adjustment for linear-time quantization and dimension segmentation for optimal bit allocation.
- We design a set of novel optimizations for SAQ that includes the multi-stage distance estimator, distance bounds, dynamic programming for bit allocation, and SIMD implementations.
- We conduct comprehensive experimental evaluation to demonstrate SAQ's superiority in accuracy (up to 5.4× improvement), quantization speed (up to 80× faster), and ANNS performance (up to 12.5x higher query throughput at 95% recall).

2 Preliminaries

In this part, we introduce the basics of RaBitQ and extended RaBitQ to facilitate the subsequent discussions.

2.1 Locally-adaptive Vector Quantization

Locally-adaptive Vector Quantization (LVQ) [1], a recently proposed vector quantization technique that efficiently compresses high-dimensional vectors while preserving the accuracy of similarity computations.

For each vector $\mathbf{x} = [x_1, \ldots, x_d]$, LVQ first mean-centers the vector by subtracting its mean μ resulting in $\mathbf{x}' = \mathbf{x} - \mu$, where $\mu = [\mu_1, ..., \mu_d]$ is the mean of all vectors in a dataset. It then computes the minimum and maximum values of the mean-centered vector, $\ell = \min_j x'_j$ and $u = \max_j x'_j$, and

divides the range $[\ell, u]$ into 2^B equal intervals, where B is the number of bits per dimension. Each coordinate x'_j is quantized to the nearest interval center:

$$Q(x'_j; B, \ell, u) = \ell + \Delta \left\lfloor \frac{x'_j - \ell}{\Delta} + \frac{1}{2} \right\rfloor, \quad \text{where } \Delta = \frac{u - \ell}{2^B - 1}. \tag{1}$$

The quantized codes for all dimensions, together with ℓ and u, are stored for each vector. LVQ is simple and efficient, and by adapting the quantization range to each vector, it can achieve better accuracy than global quantization under the same bit budget. However, its accuracy may degrade under high compression rates due to the limited dynamic range per vector.

2.2 RaBitQ

RaBitQ [18] compresses a D-dimensional vector x into a D-bit string and two floating point numbers. It provides an unbiased estimator for squared Euclidean distance and supports efficient distance computation with bit operations.

Quantization procedure. Given a data vector $\mathbf{o}_r$ and a query vector $\mathbf{q}_r$, RaBitQ first normalizes them based on a reference vector $\mathbf{c}$ (e.g., the centroid of the dataset or a cluster). The normalized vectors are expressed as

$$\mathbf{o} := \frac{\mathbf{o}_r - \mathbf{c}}{\|\mathbf{o}_r - \mathbf{c}\|}, \quad \mathbf{q} := \frac{\mathbf{q}_r - \mathbf{c}}{\|\mathbf{q}_r - \mathbf{c}\|}. \tag{2}$$

RaBitQ estimates the inner product between the normalized vectors (i.e., $\langle \mathbf{o}, \mathbf{q} \rangle$) and computes the Euclidean distance between the original vectors (i.e., $\mathbf{o}_r$ and $\mathbf{q}_r$) as:

$$\begin{aligned} &\|\mathbf{o}_r - \mathbf{q}_r\|^2 = \|(\mathbf{o}_r - \mathbf{c}) - (\mathbf{q}_r - \mathbf{c})\|^2 \\ =&\|\mathbf{o}_r - \mathbf{c}\|^2 + \|\mathbf{q}_r - \mathbf{c}\|^2 - 2 \cdot \|\mathbf{o}_r - \mathbf{c}\| \cdot \|\mathbf{q}_r - \mathbf{c}\| \cdot \langle \mathbf{q}, \mathbf{o} \rangle, \end{aligned} \tag{3}$$

where $\|\mathbf{o}_r - \mathbf{c}\|$ and $\|\mathbf{q}_r - \mathbf{c}\|$ are the distances of the data and query vectors to the reference vector $\mathbf{c}$, which can be precomputed prior to distance computation and reused. As such, RaBitQ quantizes the normalized data vector $\mathbf{o}$ and focuses on estimating $\langle \mathbf{q}, \mathbf{o} \rangle$.

To conduct quantization, RaBitQ first generates a random orthonormal matrix P and projects the normalized data vector $\mathbf{o}$ by P, [2] that is, $\mathbf{o}_p := P \cdot \mathbf{o}$. Since P is orthonormal, it preserves inner product, i.e., $\langle \mathbf{q}_p, \mathbf{o}_p \rangle = \langle \mathbf{q}, \mathbf{o} \rangle$. As such, we can estimate compress $\mathbf{o}_p$ and estimate $\langle \mathbf{q}_p, \mathbf{o}_p \rangle$. Considering that $\mathbf{o}_p$ has unit norm, RaBitQ uses the following codebook C

$$C := \left\{ +\frac{1}{\sqrt{D}}, -\frac{1}{\sqrt{D}} \right\}^D, \tag{4}$$

where each codeword is also a unit vector, and $\mathbf{o}_p$ is quantized to its nearest codeword in C. This essentially becomes selecting between -1 and +1 according to the sign for each dimension of $\mathbf{o}_p$. This is also natural since after random orthonormal projection, the dimensions of $\mathbf{o}_p$ follow the same distribution, and the same number of bits should be used to quantize each dimension. The quantized vector is denoted as $\bar{\mathbf{o}}_p$.

Distance estimator. To estimate distance, RaBitQ first rotates the normalized query vector with the random orthonormal matrix P, that is, $\mathbf{q}_p := P \cdot \frac{\mathbf{q}_r - \mathbf{c}}{\|\mathbf{q}_r - \mathbf{c}\|}$. Then, it estimates $\langle \mathbf{o}_p, \mathbf{q}_p \rangle$ as

$$\langle \mathbf{o}_p, \mathbf{q}_p \rangle = \frac{\langle \bar{\mathbf{o}}_p, \mathbf{q}_p \rangle}{\langle \bar{\mathbf{o}}_p, \mathbf{o}_p \rangle}, \tag{5}$$

[2] RaBitQ actually projects $\mathbf{o}$ by P^{-1} but this is equivalent since P^{-1} is also a random orthonormal matrix.

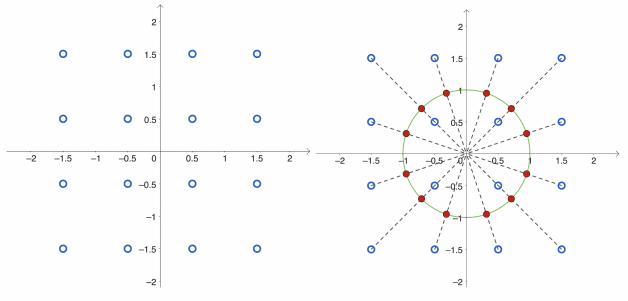


Fig. 3. The codebook structure of extended RaBitQ with dimension $D=2$ and $B=2$ bits for each dimension. Red points are the final codewords, Figure reproduced from [16].

where $\bar{\mathbf{o}}_p$ is the quantized data vector, $\langle \bar{\mathbf{o}}_p, \mathbf{q}_p \rangle$ is computed at query time, and $\langle \bar{\mathbf{o}}_p, \mathbf{o}_p \rangle$ is computed offline and stored for each vector. It has been shown that the estimator is unbiased and have tight error bound. We restate the lemmas as follows.

Lemma 2.1 (Estimator and Error Bound). *The estimator of inner product is unbiased because :*

$$\mathbb{E}\left(\langle \mathbf{o}_p, \mathbf{q}_p \rangle\right) = \frac{\langle \bar{\mathbf{o}}_p, \mathbf{q}_p \rangle}{\langle \bar{\mathbf{o}}_p, \mathbf{o}_p \rangle}. \tag{6}$$

With a probability of at least $1 - \exp(-c_0\epsilon_0^2)$*, the error bound of the estimator satisfies*

$$\mathbb{P}\left\{\left|\frac{\langle \bar{\mathbf{o}}_p, \mathbf{q}'_p \rangle}{\langle \bar{\mathbf{o}}_p, \mathbf{o}'_p \rangle} - \langle \mathbf{o}'_p, \mathbf{q}'_p \rangle\right| > \sqrt{\frac{1 - \langle \bar{\mathbf{o}}_p, \mathbf{o}'_p \rangle^2}{\langle \bar{\mathbf{o}}_p, \mathbf{o}'_p \rangle^2}} \cdot \frac{\epsilon_0}{\sqrt{D-1}}\right\} \leq 2e^{-c_0\epsilon_0^2} \tag{7}$$

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

It is also shown that $\langle \bar{\mathbf{o}}_p, \mathbf{o}_p \rangle$ is highly concentrated around 0.8, and the estimation error is smaller than $O(1/\sqrt{D})$ with high probability [2].

2.3 Extended RaBitQ

RaBitQ only allows to use 1-bit for each vector dimension, which limits accuracy. To improve accuracy, it pads $\mathbf{o}$ with zeros to a dimension $D' > D$ and uses D bits for quantization. However, this is shown to be sub-optimal, and extended RaBitQ [16] (denoted as E-RaBitQ) is proposed to support B-bit quantization for each vector dimension, where B is a positive integer. In particular, E-RaBitQ quantizes a D-dimensional vector $\mathbf{o}_r$ into a $(D * B)$-bit string and two floating point numbers.

The normalization and projection of E-RaBitQ are the same as RaBitQ but E-RaBitQ uses the following codebook $\mathcal{G}_r$ to quantize $\mathbf{o}_p$

$$\begin{aligned} \mathcal{G} &:= \left\{ -\frac{2^B - 1}{2} + u \,\middle|\, u = 0, 1, 2, 3, ..., 2^B - 1 \right\}^D \\ \mathcal{G}_r &:= \left\{ \frac{\mathbf{y}}{\|\mathbf{y}\|} \,\middle|\, \mathbf{y} \in G \right\}^D. \end{aligned} \tag{8}$$

As shown in Figure 3, the raw codewords form a regular grid in D-dimensional space, while the actual codewords are scaled to the unit sphere to have unit norm. This is because $\mathbf{o}_p$ also has unit norm.

Table 1. Frequently used notations in the paper

Notation	Definition
$\mathbf{o}_r$, $\mathbf{q}_r$	the original data and query vectors
$\mathbf{o}$, $\mathbf{q}$	the rotated and deducted data and query vectors
$\bar{\mathbf{o}}$	the quantized vector of $\mathbf{o}$
$\bar{\mathbf{o}}_a$	the adjusted quantized vector of $\mathbf{o}$
$\bar{\mathbf{c}}$, $\bar{\mathbf{c}}_a$	the quantization code of $\bar{\mathbf{o}}$ and $\bar{\mathbf{o}}_a$
$[C]$	an integer set with $\{0, 1, 2, \cdots, C\}$

A data vector $\mathbf{o}_p$ finds the nearest codeword in $\mathcal{G}_r$ as its quantized vector and stores the corresponding quantization code $\bar{\mathbf{o}}_b \in \left\{0, 1, 2, 3, ..., 2^B - 1\right\}^D$ for the codeword. However, it is difficult to find the neatest codeword in $\mathcal{G}_r$. E-RaBitQ propose a pruned enumeration algorithm to search the codeword with a time complexity of $O(2^B \cdot D \log D)$. As such, the quantization time can be long with using a few bits (e.g. $B \geq 8$) for each dimension for high accuracy.

The distance estimator of E-RaBitQ is the same as RaBitQ, and its error bound is shown empirically as follows.

Remark 1 (Error Bound). *Let ϵ be the absolute error in estimating the inner product of unit vectors. With >99.9% probability, we have $\epsilon < 2^{-B} \cdot c_\epsilon / \sqrt{D}$ where $c_\epsilon = 5.75$*

This is asymptotically optimal as the error scales with 2^{-B}.

It has been shown that when $B > 1$, taking the most significant bit for each dimension can get the quantized vector of the original RaBitQ. As such, E-RaBitQ proposes a progressive strategy to accelerate vector search, which first uses the most significant bits of $\bar{\mathbf{o}}_p$ to compute bounds for the distance, and full-precession computation is used only when $\mathbf{o}$ is likely to have smaller distance than the current top-k neighbors. However, taking $b < B$ but $b > 1$ bits from $\bar{\mathbf{o}}_p$ does not necessary form a valid quantized vector. In the subsequent paper, we denote E-RaBitQ as RaBitQ for simplicity.

3 Code Adjustment

As discussed in Section 2.3, RaBitQ has a complexity of $O(2^B \cdot D \log D)$ for encoding a D-dimension vector with $B \cdot D$ bits. Therefore, vector quantization can be time consuming. For example, for a dataset with a billion vectors, $D = 3072$ and $B = 9$, RaBitQ takes more than 3,600 CPU hours. As we mentioned in Section 1, SAQ segments dimensions and uses more bits for segments with large magnitudes. If we directly use RaBitQ for each segment, it could lead to a longer quantization time since $B > 10$ bits may be used for each dimension of the leading segments and RaBitQ's indexing time grows exponentially with B.

To reduce quantization time, we propose *code adjustment quantization* (*CAQ*), which reduces the quantization complexity drastically from $O(2^B \cdot D \log D)$ to $O(D)$, while maintaining the same empirical error and efficiency for distance estimation as RaBitQ. Another special feature of CAQ is, using B bits for each dimension, taking first b bits ($1 < b < B$) for each dimension from the quantized code $\bar{\mathbf{c}}_a$ still forms a valid quantized vector. This provides more opportunities for progressive distance computation, which is not supported in RaBitQ. Table 1 lists the frequently used notations.

Insights. RaBitQ has high quantization complexity because it requires the quantized vector (and thus vector codewords) $\bar{\mathbf{o}}_p$ to have unit norm. This makes the dimensions of $\bar{\mathbf{o}}_p$ dependent as their

squared values must sum to 1 and prevents handling each dimension of $\mathbf{o}_p$ independently. Such a design is reasonable at first glance since the norm $\|\mathbf{o}_r - \mathbf{c}\|$ is stored explicitly, and $\bar{\mathbf{o}}_p$ quantizes the direction vector $\mathbf{o}_p$, which should have unit norm. However, by inspecting the estimator in Eq (5), we observe that the norm of $\bar{\mathbf{o}}_p$ does not affect estimation. This is, $\bar{\mathbf{o}}_p$ appears in both the numerator and denominator, and thus scaling $\bar{\mathbf{o}}_p$ will not change the estimated value. Instead, $\bar{\mathbf{o}}_p$ should only be required to align with $\mathbf{o}_p$ in direction. Therefore, we remove the unit norm constraint for $\bar{\mathbf{o}}_p$ and consider each dimension of $\mathbf{o}_p$ individually. Intuitively, CAQ works like coordinate descent in optimization, i.e., it starts with a raw quantized vector and then adjusts its dimensions to better align with $\mathbf{o}_p$ in direction.

3.1 Quantization Procedure

Like RaBitQ, CAQ deducts a reference vector $\mathbf{c}$ from the data and query vectors and rotates them with a random orthonormal matrix P. Let $\mathbf{o}_r$ and $\mathbf{q}_r$ be the raw data and the query vectors, and the rotated vectors are:

$$\mathbf{o} = P \cdot (\mathbf{o}_r - \mathbf{c}), \quad \mathbf{q} = P \cdot (\mathbf{q}_r - \mathbf{c}). \tag{9}$$

The distance between the original data vector $\mathbf{o}_r$ and query vector $\mathbf{q}_r$ is equal to the distance between the rotated data vector $\mathbf{o}$ and query vector $\mathbf{q}$ since P is orthonormal. For simplicity, we also call $\mathbf{o}$ and $\mathbf{q}$ data and query vectors.

CAQ initializes the quantized vector $\bar{\mathbf{o}}$ similar with LVQ [1]. In particular, for data vector $\mathbf{o}$, let $v_{\max} = \max_{i\in[D]} |\mathbf{o}[i]|$, i.e., the maximum magnitude in $\mathbf{o}$. We divide the range of $[-v_{\max}, v_{\max}]$ into 2^B uniform intervals, each with length $\Delta = (2 \cdot v_{\max})/2^B$. The midpoint of the x^{th} interval is $-v_{max} + \Delta(x + 0.5)$, which is used to quantify the dimensions of $\mathbf{o}$ whose value falls in its corresponding interval. These midpoints form a D-dimensional uniform grid, and for each dimension $i \in [D]$, we have

$$\bar{\mathbf{c}}[i] := \left\lfloor \frac{\mathbf{o}[i] + v_{\max}}{\Delta} \right\rfloor \tag{10}$$

as the quantization code. Data vector $\mathbf{o}$ is quantized as

$$\bar{\mathbf{o}} := \Delta(\bar{\mathbf{c}} + 0.5) - v_{\max} \cdot \mathbf{1}_D. \tag{11}$$

LVQ stops here and uses $\bar{\mathbf{o}}$ to estimate distance. However, according to RaBitQ's analysis [16], to approximate the data vector, the quantization vector should align with the data vector $\mathbf{o}$ in

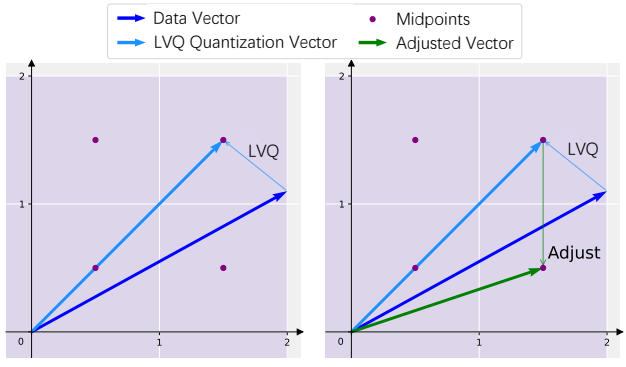


Fig. 4. The procedure of CAQ for quantization, which first starts with LVQ and then adjusts the quantized vector to align with the data vector in direction.

Algorithm 1: Adjustment of the LVQ Quantized Vector

1 **Input**: round limit r, data vector $\mathbf{o}$, start point $\bar{\mathbf{o}}$, value $v_{\max}$
2 **Output**: the final quantized vector $\bar{\mathbf{o}}_a$ for data vector $\mathbf{o}$
3 Step size $\Delta \leftarrow (2 \cdot v_{\max})/2^B$;
4 Initialize $\mathbf{x} \leftarrow \bar{\mathbf{o}}$;
5 **for** *round* $\leftarrow 1$ **to** r **do**
6 **for** $i \leftarrow 1$ **to** D **do**
7 **for** $\delta \in \{\Delta, -\Delta\}$ **do**
8 $\mathbf{x}' \leftarrow \mathbf{x}$;
9 $\mathbf{x}'[i] \leftarrow \mathbf{x}'[i] + \delta$;
10 **if** $\mathbf{x}'[i] \in [-v_{\max}, v_{\max}] \wedge \mathcal{L}(\mathbf{x}', \mathbf{o}) > \mathcal{L}(\mathbf{x}, \mathbf{o})$ **then**
11 $\mathbf{x} \leftarrow \mathbf{x}'$;
12 $\bar{\mathbf{o}}_a \leftarrow \mathbf{x}$
13 **return** $\bar{\mathbf{o}}_a$

direction. As such, quantization should find the vector $\mathbf{x}$ with the largest cosine similarity to $\mathbf{o}$, which is defined as

$$\mathcal{L}(\mathbf{x}, \mathbf{o}) = \frac{\mathbf{x} \cdot \mathbf{o}}{\|\mathbf{x}\|_2 \cdot \|\mathbf{o}\|_2}. \tag{12}$$

LVQ only produce the quantization vector $\bar{o}$ that is the nearest to the data vector $\mathbf{o}$ among all possible quantization vectors and it may not optimize the objection function in Eq. (12). However, as we have observed, the cosine similarity can be significantly improved by adjusting and refining $\bar{\mathbf{o}}$ to better align with the data vector $\mathbf{o}$. We propose an efficient code adjustment algorithm shown in Algorithm 1. Lines 7-11 try to adjust a dimension of the quantized vector by a step of Δ and see if the direction is better aligned. The adjustment iterates over all dimensions and for a limited number of rounds (Lines 5-6).

Figure 4 illustrates how the code adjustment algorithm adjusts the initial code produced from LVQ to improve cosine similarity.

Since Algorithm 1 only changes one vector dimension for each adjustment, we do not need to recompute $\mathcal{L}(\mathbf{x}', \mathbf{o})$ by enumerating all dimensions. Instead, we only need to recompute the contribution of the current dimension to $\mathcal{L}(\mathbf{x}', \mathbf{o})$. Thus, the time complexity of each adjustment is $O(1)$. The overall time complexity of Algorithm 1 is $O(r \cdot D)$, which is in the same $O(D)$ order for computing $\bar{\mathbf{o}}$. We evaluate the quantization accuracy with different number of adjustment iteration r in Section 5.2. In practice, we recommend setting $r \in [4, 8]$, which is sufficient to obtain a quantized vector with high quality.

After adjustment, we obtain the approximate quantization vector $\bar{\mathbf{o}}_a$ and compute the final quantization code $\bar{\mathbf{c}}_a$ using Eq. (10). The final quantization code is a D-dimensional vector whose coordinates are B-bit unsigned integers so that we can store the code with a $(B * D)$-bit string. Like RaBitQ, CAQ uses two additonal float numbers for each vector to store the norm and $\langle \bar{\mathbf{o}}_a, \mathbf{o} \rangle$.

3.2 Distance Estimation

We adopt the same distance estimator as RaBitQ, i.e., approximating $\langle \mathbf{o}, \mathbf{q} \rangle$ as $\langle \bar{\mathbf{o}}_a, \mathbf{q} \rangle / \langle \bar{\mathbf{o}}_a, \mathbf{o} \rangle \cdot |\mathbf{o}|^2$. As discussed earlier, $\langle \bar{\mathbf{o}}_a, \mathbf{o} \rangle$ is precomputed and stored. In the query phase, we can compute $\langle \bar{\mathbf{o}}_a, \mathbf{q} \rangle$ using the integer quantization code $\bar{\mathbf{c}}_a$ without decompression as follows

$$\begin{aligned}\langle \bar{\mathbf{o}}_a, \mathbf{q} \rangle &= \sum_{i=1}^{D} \bar{\mathbf{o}}_a[i] \cdot \mathbf{q}[i] \\ &= \sum_{i=1}^{D} (\Delta(\bar{\mathbf{c}}_a[i] + 0.5) - v_{\max}) \cdot \mathbf{q}[i] \\ &= \Delta \langle \bar{\mathbf{c}}_a, \mathbf{q} \rangle + q_{\text{sum}}(-v_{\max} + \Delta/2),\end{aligned} \tag{13}$$

where $q_{sum} = \sum_{i=1}^{D} \mathbf{q}[i]$, which only needs to be computed once for each query vector. Like RaBitQ, if we quantize each dimension of a query vector to integer, Eq. 13 will mostly use integer computation.

Progressive distance approximation. CAQ supports progressive distance approximation using the prefix of quantized code of each dimension with an arbitrary length. In particular, let $\bar{\mathbf{c}}$ denote the first b bits sampled from the native B bit quantization code $\bar{\mathbf{c}}_a$ in each dimension, that is, $\bar{\mathbf{c}}_s = \lfloor \bar{\mathbf{c}}_a / 2^{B-b} \rfloor$. The inner product can be estimated using $\bar{\mathbf{c}}_s$ via Eq. 13 by replacing Δ with $\Delta' = \Delta \cdot 2^{B-b}$ and $\bar{\mathbf{c}}_a$ with $\bar{\mathbf{c}}_s$. Our experiments in Section 5.2 show that the prefix $\bar{\mathbf{c}}_s$ yields almost the same estimation error as a native CAQ quantized vector using b-bits for each dimension. The progressive distance estimator of CAQ enables vector search to conduct progressive distance refinement (e.g., first with 1-bit, then 2-bit, and finally 8-bit). This allows us to provide multiple efficiency-accuracy trade-off options from the same quantization code to satisfy various requirements (e.g., we use it in our multi-stage estimation in Section 4.3). In contrast, RaBitQ only supports progressive approximation with $b = 1$, which is rather restrictive in its usage.

3.3 Analysis

Recall that, to approximate a data vector $\mathbf{o}$, RaBitQ finds its nearest codeword in codebook $\mathcal{G}_r$ that contains unit norm vectors. That is, RaBitQ solves the following optimization problem for quantization

$$\text{argmax}_{\mathbf{x} \in \mathcal{G}_r} \mathcal{L}(\mathbf{x}, \mathbf{o}). \tag{14}$$

CAQ finds the codeword that best aligns with $\mathbf{o}$ in direction in a codebook $\mathcal{D}_{\text{CAQ}}$. That is, CAQ solves the following optimization problem:

$$\text{argmax}_{\mathbf{x} \in \mathcal{D}_{\text{CAQ}}} \mathcal{L}(\mathbf{x}, \mathbf{o}) \tag{15}$$

among $\mathcal{D}_{\text{CAQ}}$. We show in Lemma 3.1 that the codebooks of RaBitQ and CAQ are essentially the same.

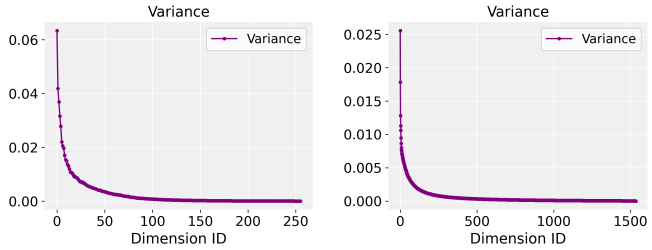
Lemma 3.1. *$\mathcal{D}_{CAQ}$ of CAQ is equivalent to $\mathcal{G}_r$ in RaBitQ.*

Proof. From Lines 6-11 of Algorithm 1, the dimensions values of the vector $\mathbf{x}$ are taken from $\left\{-v_{\max} + \Delta \cdot (i + 0.5) \mid i \in [2^B - 1]\right\}$. Thus, we have

$$\mathcal{D}_{\text{CAQ}} = \left\{-v_{\max} + \Delta \cdot (i + 0.5) \mid i \in [2^B - 1]\right\}^D.$$

With $\Delta = 2 \cdot v_{\max} / 2^B$, dividing the vectors in $\mathcal{D}_{\text{CAQ}}$ by $v_{\max}$ gives $\left\{-(2^B - 1)/2 + i \,\middle|\, i \in [2^B - 1]\right\}^D$, which matches the unnormalized codebook $\mathcal{G}$ of RaBitQ in Eq. 8. The normalization only changes the norm of the codewords but not the direction, and we have discussed that the norms of the codewords do not affect the estimated inner product. □

Therefore, if CAQ can solve its optimization problem in Eq (15), it achieves the unbiased estimation and error bound as RaBitQ. Although the coordinate descent style optimization of Algorithm 1 does not necessarily converge to the optimal, empirically, we observe that CAQ achieves identical estimation errors as RaBitQ.



(a) DEEP (b) OpenAI-1536

Fig. 5. Variance of vector dimension after PCA projection.

4 Dimension Segmentation

In this section, we present *Segmented CAQ* (*SAQ*). SAQ combines the benefits of both *dimension balancing* and *dimension reduction* to push the performance limits of vector quantization. In Section 4.1, we first introduce the motivation of SAQ and formulate the problem of finding a quantization plan. Then we present a dynamic programming algorithm to find the optimal quantization plan in Section 4.2. For query processing, in Section 4.3, we show how to use the quantization plan to estimate distance. We also present a multi-stage estimator that facilitates candidate pruning based on the property of the quantization plan.

4.1 Motivation and Problem Formulation

CAQ requires the random orthonormal projection because it treats each vector dimension equivalently by quantizing them with the same number of bits. If some dimensions have much larger variances than the others, their accuracy will hurt. In an extreme example, consider a dataset of two-dimensional vectors whose values for the first dimension follow the normal distribution and the values for the second dimension are the same fixed value. The bits assigned for the second dimension will be wasted as we can store one copy of the specific value for all data vectors. By a random projection, the variance is scattered across all dimensions to ensure that all bits are fully utilized to boost accuracy. We refer to this technique as **dimension balancing**.

In the reverse direction, if we concentrate the data vectors' variances into certain dimensions, we can remove the dimensions with negligible variances. PCA is widely used for this purpose. A recent study [54] indicates that the high-dimensional vector coordinates, once rotated by a PCA matrix, exhibit a long-tailed variance distribution, which implies that only a few dimensions hold most of the variance (Figure 5). This characteristic was used to create an approximate distance estimator that uses the leading dimensions. These techniques are commonly known as **dimension reduction**. Dimension reduction is less general, as it is highly dependent on the data distribution. When PCA fails to polarize the variance, a fixed dimension reduction ratio will result in poor accuracy.

The common idea behind these two methods indicates that dimensions with higher variance should be allocated more bits and those with lower variance fewer bits. Building on this observation, we introduce *Segmented CAQ* (*SAQ*), which uses varying compression ratios based on dimension variance to improve accuracy. Unlike a common parameter B for all dimensions as in CAQ, SAQ takes a parameter Q_{quota} to represent the total bit quota to quantize the entire vector. Given a dataset, we first learn a PCA rotation matrix and rotate all data vectors with the PCA matrix. In this way, we polarize the variance and obtain $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_D$, where σ_i is the variance of the values

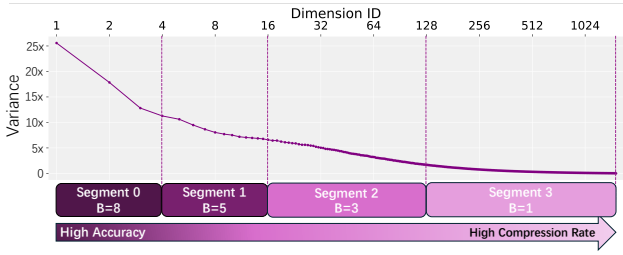


Fig. 6. An illustration of dimension segmentation.

taken by the i^{th} dimensions of the rotated vectors. SAQ then performs quantization following a quantization plan, and an example is illustrated in Figure 6.

To define a quantization plan, we first introduce the concept of a dimension segment Seg, which is a set of continuous vector dimensions. For a dimension segment Seg and a vector $x \in X$, $x[Seg]$ indicates the projection of x onto the dimensions in Seg, and $X[Seg]$ represents the set of all such projected vectors. We further define a tuple (Seg, B) that indicates that SAQ will quantize each dimension in Seg using B bits. A quantization plan is given by a set of such tuples $\mathcal{P} := \{(Seg_1, B_1), (Seg_2, B_2), ...\}$, where Seg_i is a dimension segment and B_i is the number of bits used to quantize the dimensions in Seg_i. The union of all these Seg_i is $[D]$. The total bit consumption of this plan is

$$C(\mathcal{P}) = \sum_{(Seg,B)\in\mathcal{P}} B \cdot |Seg|. \tag{16}$$

We want to find a quantization plan $\mathcal{P}$ that achieves a low estimation error with $C(\mathcal{P}) \le Q_{quota}$. The number of possible quantization plans can be prohibitively large, and it is difficult to predict the estimation error without deploying and testing the plan with real queries. To this end, we provide an efficient way to model the estimation error of a quantization plan and a method to search for good quantization plan. Note that when quantizing each segment with CAQ, we still apply the random orthonormal projection beforehand such that the dimensions are similar in variance.

4.2 Quantization Plan Construction

According to our experiments and the analysis of RaBitQ [16], the relative error of quantization a segment (or vector) is proportional to 2^{-B}, where B is the number of bits used to quantify the segment. This is natural because with 1 more bit, we can reduces the quantization resolution by $1/2$. Moreover, we observe that, after PCA projection, the value distribution of a dimension almost follows a normal distribution with a mean close to 0, as illustrated in Figure 7. Therefore, we propose the following expression to model the error introduced in the index phase:

$$\text{ERROR}(Seg, B) = \sum_{i\in Seg} \frac{\mathbb{E}\left[|\mathbf{o}_r[i] \cdot \mathbf{q}_r[i]|\right]}{2^{B+1}} = \frac{1}{2^B\pi} \sum_{i\in Seg} \sigma_i^2, \tag{17}$$

where $\mathbf{o}_r$ and $\mathbf{q}_r$ are the raw data vector and raw query vector, and σ_i means the value variance of the i-th dimension[3]. The error introduced by a quantization plan $\mathcal{P}$ can then be modeled as

$$\text{ERROR}(\mathcal{P}) = \sum_{(Seg,B)\in\mathcal{P}} \text{ERROR}(Seg, B). \tag{18}$$

[3]Here, we assume query vectors share the same distribution with data vectors. In fact, we can drop the normal distribution assumption and simply use the empirical variance of each dimension (i.e., σ_i^2). This will remove the π in Eq (17).

Algorithm 2: Quantization Plan Search

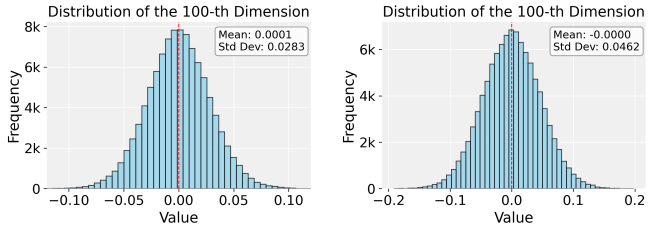
```
1  Input: vector dimension D, total bit quota Q_quota
2  Output: Quantization plan P.
3  p_{0,0} ← (∅, ∅)
4  for d ← 0 to D − 1 do
5      for Q ← 0 to Q_quota do
6          if p_{d,Q} exists then
7              for d' ← d to D do
8                  seg ← {d + 1, d + 2, ..., d'}
9                  for b' is valid quantization bits do
10                     Q' ← Q + b' * d'
11                     p' ← p_{d,Q} ∪ (seg, b')
12                     if p_{d',Q'} not exist or ERROR(p_{d',Q'}) > ERROR(p') then
13                         p_{d',Q'} ← p'
14 P ← ∅
15 for Q ← 0 to Q_quota do
16     if p_{D,Q} exists
17     JAMES CHENG[4], The Chinese University of Hong Kong, Hong Kong SAR
18     ERROR(p_{D,Q}) < ERROR(P) then
19         P ← p_{D,Q}
20 return P
```

The problem then becomes finding a plan under the given bit quota Q_{quota} with minimal ERROR($\mathcal{P}$). To solve this problem, we devise a dynamic programming algorithm as presented in Algorithm 2.

Let $p_{d,Q}$ denote the quantization plan that contains the first d dimensions and uses Q bits quota in total. We use dynamic programming to find the optimal quantization plan. Specifically, each time we enumerate an existing quantization plan $p_{d,Q}$ (lines 4-6) and append a new segment *seg* that contains a set of dimensions $\{d+1, d+2, ..., d'\}$ and use b' bits to quantize each dimension (lines 7-11). We create this new quantization plan or update it if it already exists and the new one is better (lines 12-13). After the whole process, we obtain the optimal quantization plan $\mathcal{P}$ that minimizes the total error within the total bit quota Q_{quota} (lines 15-17).

The time complexity of this dynamic programming is $O(D^2 \cdot Q_{quota})$, which is insignificant compared to the time complexity of quantization as it does not loop over vectors. In practice, we set the size of each segment to be a multiple of 64 to align with cache line size. Moreover, as each segment has some extra computation overhead for the distance estimator, which we will shown soon, using many small segments is not efficient. Thus, we choose the quantization plan that gives an error close to (e.g., <0.1% of) the minimum but with the least number of segments. On all our experimented datasets, quantization plan search can finish within 1 second.

Figure 6 illustrates an example of our quantization plan (disregarding the 64 limits for simplicity). We allocate more bits to dimensions with high variance to attain higher precision, while applying higher compression rates to dimensions with low variance to maintain the total bit budget. After



(a) DEEP (b) OpenAI-1536

Fig. 7. The value distribution at two sampled dimensions after PCA projection, all vectors are considered.

constructing the quantization plan, we use it to split the dimensions of data vectors into segments and quantize each segment separately just as a normal vector.

Dimension segmentation relies on a skewed eigen value distribution over the dimensions such that more bits can be used for dimension segments with larger eigen values. When the eigen value distribution is perfectly uniform, the chance for improvement varnishes. However, for these extreme cases, the quantization plan will contain only one segment that contains all dimensions and will still match the performance of CAQ.

4.3 Multi-Stage Distance Estimation

To process an ANN query, we also split the query vector into segments, estimate the inner product of each segment correspondingly, and combine them to obtain the final distance. Instead of trivially summing up the inner product of each segment, we can also use the feature of the quantization plan to improve the estimator.

Recent studies [17] show that reliably identifying the nearest neighbor (NN) does not require exact distance calculations for all candidate vectors. Instead, approximate distance estimates or distance bounds can filter out most unlikely candidates (e.g. if a lower bound exceeds the current best NN distance). In Section 3.2, we introduce a progressive distance approximation method that supports progressive refinement for CAQ. For SAQ, we further improve the estimator by leveraging the quantization plan. In particular, the quantization plan partitions the dimensions into segments, and segments with high variance contribute more to the final distance. This inspires us to design a multi-stage estimator that gradually refines the distance estimation by starting with the high-variance segments and gradually adding more segments. Once the distance exceeds the current best NN distance, we can prune the vector.

Moreover, we can even obtain a lower bound of the estimated distance with almost no computation. Specifically, we can predict the distance contribution of each segment with the value distributions of the dimensions in the segment.

For a data vector $\mathbf{o}_r$, the contribution of the dimensions inside segment Seg to the final inner product can be written as

$$\langle \mathbf{o}_{seg}, \mathbf{q}_{seg} \rangle = \sum_{i \in Seg} \mathbf{q}[i] \cdot \mathbf{o}[i]. \tag{19}$$

According to our observation, after PCA projection, the value distribution of a dimension almost follows a normal distribution with a mean close to 0, as illustrated in Figure 7. The variance of expression above can be written as:

$$\sigma^2_{Seg} = Var\left(\langle \mathbf{o}_{seg}, \mathbf{q}_{seg} \rangle\right) = \sum_{i \in Seg} \mathbf{q}^2[i] \cdot \sigma_i^2, \tag{20}$$

where σ_i^2 is the variance of dimension i among the dataset.

According to Chebyshev's inequality, we have:

$$\mathbb{P}\left(\left|\langle \mathbf{o}_{seg}, \mathbf{q}_{seg}\rangle\right| \geq \mathrm{Est}_v(Seg)\right) \leq \frac{1}{m^2}, \tag{21}$$

where $\mathrm{Est}_v(Seg) = \sigma_{Seg} \cdot m$ and m is a predefined constant. We can use $\mathrm{Est}_v(Seg)$ to predict a lower bound of the contribution of each stage to the final inner product with confidence $1 - \frac{1}{m^2}$.

It is worth noting that σ_{Seg} only needs to be computed once for each query and is shared by all candidate vectors in the query phase. We combine the new estimator $\mathrm{Est}_v(Seg)$ with the one in CAQ to conduct a multi-stage estimator, which is a more fine-grained progressive distance refinement provide by SAQ. The multi-stage estimator significantly reduces unnecessary computation and memory access. Our experiments show that the average bit access of the multi-stage estimator is even smaller than the dimension of the dataset, which means that the multi-stage estimator can prune most of the candidates without computing all the segments.

5 EXPERIMENTAL Evaluation

Baselines. We compared CAQ and SAQ with four other quantization methods listed below.

- **Extended RaBitQ [16]:** The state-of-the-art quantization method, which quantizes the directions of vectors and stores the lengths. We call it RaBitQ in the experiments.
- **LVQ [1]:** A recent vector quantization method that achieves good accuracy. LVQ first collects the minimum value v_l and maximum value v_r of all coordinates for a vector. Then it uniformly divides the range $[v_l, v_r]$ into $2^B - 1$ segments. The floating point value of each coordinate of that vector is rounded to the nearest boundary of the intervals and stored as a B-bit integer.
- **PCA:** We implemented a simple PCA method that first projects all data vectors with a PCA matrix and then quantizes the projected vectors by dropping the insignificant dimensions directly. The dropping rate is equal to the compressing rate.
- **PQ [24]:** A popular quantization method that is generally used with a high compression rate and is widely used in industry. We use the Faiss PQ implementation [12]. PQ supports two settings $nbits = 4$ and $nbits = 8$, which is the number of bits used to represent each sub-vector after quantization. According to [16], $nbits = 8$ produces consistently better than $nbits = 4$, so we report the result under this setting.

We do not compare with OPQ [19], LSQ [33], and scalar quantization (SQ) because they are outperformed by RaBitQ. To test the performance of the vector quantization methods for ANNS, we build IVF index [23] for all datasets following RaBitQ. In particular, IVF groups the vectors into clusters and scans the top-ranking clusters for each query. Following the common setting of IVF, we used 4,096 clusters for the datasets. We are aware that proximity graph indexes [14, 32] are also popular, but combining with vector indexes is not our focus and we leave it to future work.

Table 2. Datasets.

Dataset	Size	D	Query Size	Type
DEEP	1,000,000	256	1,000	Image
GIST	1,000,000	960	1,000	Image
MSMARC	10,000,000	1024	1,000	Text
OpenAI-1536	999,000	1536	1,000	Text

All methods were optimized with the SIMD instructions with AVX512 to ensure a fair efficiency comparison, and the performance improvements of SAQ generalize across platforms.

Datasets. We used four public real-world datasets that have various dimensionalities and data types, as shown in Table 2. These datasets include widely adopted benchmarks for the evaluation of ANN algorithms[4, 16, 18, 28, 54] (DEEP and GIST[5]) and embeddings generated by language models. The MSMARCO[6] dataset contains the embeddings for the TREC-RAG Corpus 2024 [48] embedded with the Cohere Embed V3 English model [8] and the OpenAI-1536[7] dataset is produced by model *text-embedding-3-large* of OpenAI [37]. We used the query sets provided by the DEEP and GIST datasets. For MSMARCO and OpenAI-1536, we randomly removed 1,000 vectors from each dataset and used them as query vectors.

Performance metrics and machine settings. For the experiments that evaluate the quantization accuracy under different space quota, we measured the accuracy of the estimator in terms of average relative error, maximum relative error, and recall. The relative error is defined as $|d^2_{\text{est}} - d^2_{\text{real}}|/d^2_{\text{real}}$, where d^2_{est} and d^2_{real} are the estimated and real squared Euclidean distance, receptively. Recall is the percentage of true nearest neighbors successfully retrieved with top $k = 100$ and $nprob = 200$. These metrics are widely adopted to measure the accuracy of ANN algorithms [4, 15, 22, 28, 38, 44]. All metrics were measured on every query and averaged across the entire query set. The compression rate was measured by the number of bits used to quantize a single dimension, which is computed by dividing the total number of bits used to quantize all the dimensions by the number of dimensions.

For the experiments that evaluate indexing, we measured the index time of different methods using 24 threads. For the experiments that evaluate the performance of ANNS, we measured QPS, i.e., the number of queries processed per second, under different average distance ratios and recalls to show the performance of each method with different compression rates. The average distance ratio is the average of the distance ratios of the retrieved nearest neighbors over the true nearest neighbors. These metrics are widely adopted to measure the performance of ANN algorithms [4, 15, 16, 35].

All the experiments were run on a server with 4 Intel Xeon Gold 6252N@ 2.30GHz CPUs (with 24 cores/48 threads each), 756GB RAM. The C++ source code is compiled by GCC 11.4.0 with -Ofast -march=native under Ubuntu 22.04 LTS container with AVX512 enabled. All the results, including indexing and search, were computed using 24 threads bound to 24 physical cores for all methods.

5.1 Main Results

Quantization accuracy. In this experiment, we evaluate the accuracy of different quantization methods at different compression rates with fixed $nprob = 200$. We focus on both high compression rate ($B = 0.2$, compressing 160× approximately) and moderate compression rate ($B = 8$, compressing 4× approximately). Note that PQ's compression rate is not controlled by B as in the other methods, but we obtained a very close compression rate for PQ as the other methods for each B in all our experiments.

As reported in Figure 8, SAQ (the blue curve) achieves consistently better average relative error and recall than all baseline methods at the same compression rate. The maximum relative error of SAQ is relatively less stable in low-dimension datasets (e.g., DEEP) due to the limitation of SIMD where the granularity of segmentation is at least 64, which leads to a suboptimal quantization plan. Nevertheless, as the dimension increases, this limitation is hidden and both the average and

[5]https://www.cse.cuhk.edu.hk/systems/hash/gqr/datasets.html
[6]https://huggingface.co/datasets/Cohere/msmarco-v2.1-embed-english-v3
[7]https://huggingface.co/datasets/Qdrant/dbpedia-entities-openai3-text-embedding-3-large-1536-1M

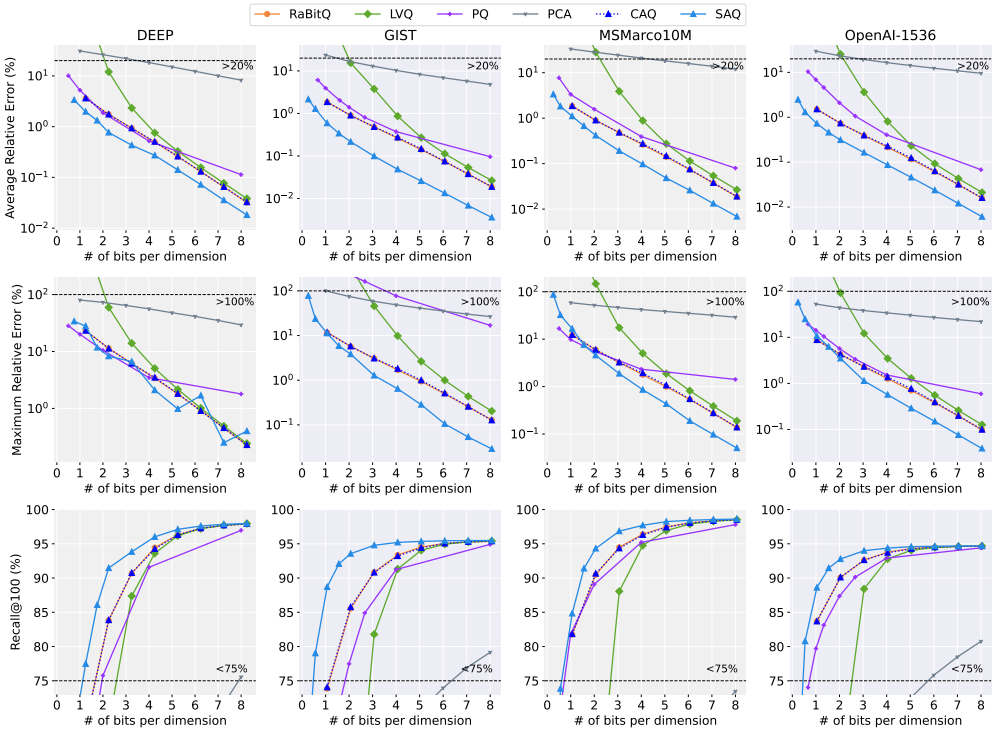


Fig. 8. Quantization accuracy of SAQ and the baselines ($nprob = 200$).

Table 3. Average relative error at B=4. SAQ reports the error, while other methods report the error blowup w.r.t. SAQ.

Method	DEEP	GIST	MSMARC	OpenAI-1536
SAQ	0.27%	0.05%	0.10%	0.09%
CAQ (ratio of SAQ)	1.9×	5.6×	2.8×	2.6×
RaBitQ (ratio of SAQ)	1.8×	5.4×	2.7×	2.5×
LVQ (ratio of SAQ)	2.8×	17.8×	9.1×	9.3×
PQ (ratio of SAQ)	1.9×	7.7×	4.1×	4.7×

maximum relative errors of SAQ decrease exponentially with the number of bits. Additionally, even our base method CAQ also consistently achieves almost the same performance as RaBitQ in this experiment, which shows that our new efficient quantization method does not compromise accuracy.

At higher compression rates ($B \leq 4$), the accuracy of the LVQ worsens dramatically as the compression rate increases. Table 3 reports the error comparison of different methods at $B = 4$. It is worth noting that as shown in Figure 8, SAQ can arealdy achieve a high recall of $> 95\%$ at $B = 4$, and in some case like for MSMARCO, even $B = 2$ is good enough.

While the maximum compression rate of RaBitQ and LVQ is around 32× (at $B = 1$), SAQ and also PQ can achieve higher compression rates with reasonable accuracy. However, for high compression

Table 4. Quantization time (in seconds) of different methods. *Speedup* is the speedup of SAQ over RaBitQ.

Method	DEEP				GIST				MSMARCO				OpenAI-1536			
	$B=1$	$B=4$	$B=8$	$B=9$	$B=1$	$B=4$	$B=8$	$B=9$	$B=1$	$B=4$	$B=8$	$B=9$	$B=1$	$B=4$	$B=8$	$B=9$
RaBitQ	0.3	3.9	21.5	41.7	2.4	16.9	83.4	165.9	33.8	178.9	897.4	1773.6	6.1	28.2	139.7	269.0
LVQ	0.3	0.3	0.3	0.4	0.8	1.1	1.3	1.4	12.6	14.8	18.5	17.9	1.5	1.8	2.3	2.3
PQ*	6.9	18.0	30.8	-	25.1	66.3	114.1	-	144.6	256.6	396.6	-	42.6	107.8	184.8	-
CAQ	0.6	1.1	0.7	0.7	2.5	5.2	3.5	2.9	28.4	60.2	32.8	31.3	6.3	10.5	6.8	7.0
SAQ	0.3	0.7	0.7	0.7	2.2	3.5	2.1	2.0	27.7	38.3	26.2	26.1	3.5	7.5	3.9	3.7
Speedup	**0.9×**	**5.9×**	**32.2×**	**59.1×**	**1.1×**	**4.8×**	**39.1×**	**84.7×**	**1.2×**	**4.7×**	**34.2×**	**67.9×**	**1.7×**	**3.8×**	**35.4×**	**73.1×**

*: The results are obtained at the same compression rate. '-' means do not support.

rates, e.g., when $B = 0.5$ ($\sim$ 64× compression), the average relative error of SAQ is significantly better than that of PQ (e.g., 4.8× lower for GIST), and impressively, is even consistently lower than the error of the state-of-the-art RaBitQ obtained at a much lower compression rate when $B = 1$ ($\sim$ 32× compression). In a more extreme case $B = 0.2$ ($\sim$ 160× compression), SAQ still achieves better accuracy than PQ. At lower compression rates ($B > 4$), SAQ also consistently achieves the lowest average and maximum relative errors and recalls compared to baselines. The average relative error is 2.5× to 5× lower than that of RaBitQ in high-dimensional datasets.

Quantization efficiency. In this experiment, we evaluate the quantization efficiency of different methods. The quantization time includes generating the random orthogonal matrix and applying the rotation to all the data vectors, but excludes the time for applying the PCA projection. Since the PCA projection and the random rotation can be combined into one matrix, this will not change the time complexity. As reported in Table 4, the quantization time of RaBitQ increases exponentially with B. When $B = 9$, RaBitQ already costs 11.8 CPU hours (i.e., 0.49 hours with 24 threads) to quantize MSMARCO, which contain 10M vectors with 1,024 dimensions. In contrast, the quantization time of CAQ and SAQ fluctuates only within a manageable time range, similar to that of LVQ. The speedup of SAQ over RaBitQ can be over 80× at $B = 9$.

ANNS performance. In this experiment, we evaluate the performance of CAQ, SAQ, and baselines to process ANNS queries. Figure 9 reports the QPS-Recall curves (upper right, i.e., higher QPS/recall, is better) and the QPS-Ratio curves (upper left, i.e., higher QPS and lower average distance ratio, is better) for both methods. Using the IVF index, as the number of clusters to probe increases, QPS decreases as computation increases, but the probability of finding the ground truth increases. A lower compression rate can produce a more accurate estimated distance to reach a higher recall, but requires more computation that slows down the distance estimator. We set $m = 2$ for the multi-stage estimator of SAQ in GIST and $m = 4$ in other datasets. We report the results of SAQ for $B = \{2, 3, 5\}$, RaBitQ for $B = \{3, 5\}$ (it does not support $B = 2$), LVQ for $B = 4$ and PQ for the compression rate 8× (equivalent to 4bit).

When the number of clusters to probe is low, LVQ can achieve high QPS, especially for the high-dimensional datasets, since it does not require the rotation of the query vectors. However, as the number of clusters to probe increases, the QPS of LVQ drops significantly due to the high computation cost of a single distance estimation and can not reach the recall as 3bit RaBitQ and SAQ, even 2bit SAQ.

For SAQ, in the low-dimensional dataset (DEEP), it can achieve a higher recall than RaBitQ at the same compression rate. In fact, 2bit SAQ attains a higher recall than 3bit RaBitQ. However, the QPS of SAQ is hindered by the additional computation of the multi-stage estimator. This is because for low-dimensional data, the computation of an individual estimator is minor and the effect of the

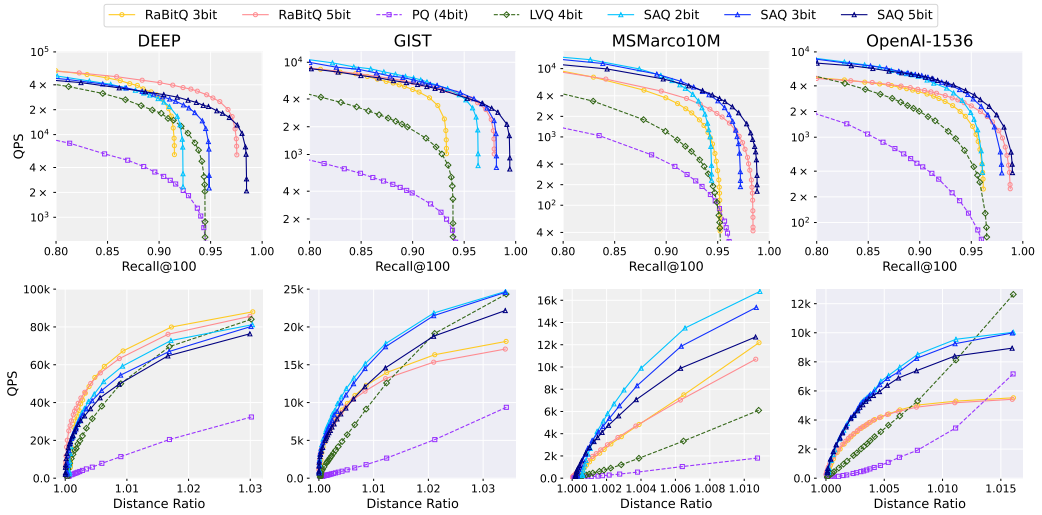


Fig. 9. The performance of SAQ and the baselines for ANNS (higher QPS/recall and lower average distance ratio are better).

Table 5. Best QPS at 95% recall for each method. '-' means cannot reach 95% recall.

Method	DEEP		GIST			MSMARCO		OpenAI-1536		
	$B=3$	$B=5$	$B=2$	$B=3$	$B=5$	$B=3$	$B=5$	$B=2$	$B=3$	$B=5$
RaBitQ	–	**27721**	–	–	**4018**	235	**1958**	–	1338	**2329**
SAQ	–*	**19407**	3915	**4683**	4073	3427	**3726**	1624	2812	**3112**

* The maximum recall of SAQ in DEEP is 94.86% when $B=3$.

early pruning of candidates is negligible, while the constant overhead of the multi-stage estimator is more significant.

The multi-stage estimator is more suitable for high-dimensional datasets, where the computation of a single estimator is heavier and the overhead of the multi-stage estimator becomes minor. Early pruning significantly reduces computation and memory access. Thus, for high-dimension datasets, SAQ consistently outperforms RaBitQ in terms of QPS, recall, and average distance ratio at the same compression rate. Even at 2bit SAQ can still almost outperform 3bit RabitQ and produces $\sim$ 95% recall.

In Table 5, we also report the QPS for different datasets and configurations when 95% recall is achieved. For GIST and OpenAI-1536, SAQ can achieve > 95% recall and reasonable QPS with $B=2$. For MSMARCO, SAQ can also produce a maximum 94.4% recall with $B=2$. Thus, $B=3$ is sufficient for SAQ to produce 95% recall for these high-dimensional datasets, and $B=5$ almost gives the best QPS. We notice that for MSMARCO, the QPS of SAQ is 14.6× that of RaBitQ when $B=3$. This significant difference occurs because RaBitQ approaches its maximum recall at $B=3$ due to error limitations, causing a sharp decline in its QPS, while SAQ maintains high performance with room for further recall improvement.

Table 6. Storage space of the quantized vectors for MSMARCO. '-': the configuration not supported.

B	0.5	1	2	4	6	8
RaBitQ (MB)	–	1363	–	5025	–	9908
CAQ (MB)	–	1379	2600	5041	7483	9924
SAQ (MB)	838	1449	2671	5114	7481	9922

Raw data vector space consumption is 39,100 MB.

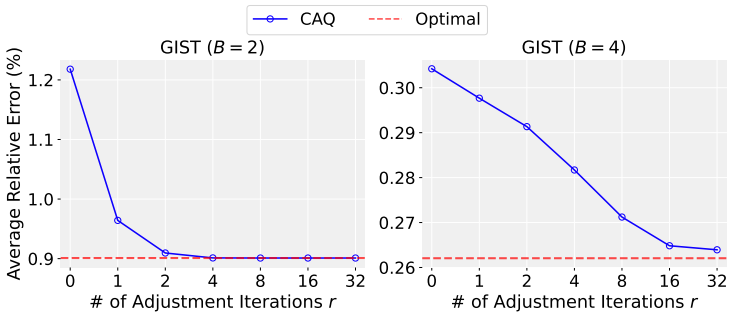


Fig. 10. Quantization accuracy with different number of code adjustment iterations r. *Optimal* means the error produced by the RabitQ quantization code, which is optimal.

5.2 Micro Results

Space consumption. We first report the space consumption of SAQ and RaBitQ with different configurations B. The space includes the quantization code, two extra factors per data vector and other factors that consume constant space. We only report the results from the MSMARCO dataset under a few configurations of B. The space consumption is proportional to the number of data vectors and the number of dimensions for the other datasets, and also proportional to other configurations of B.

As reported in Table6, the space consumption is almost proportional to B. We note that, due to the constant overhead of some factors, the compression rate may not be as expected in small B. The space consumption of SAQ is slightly larger than the baseline since the multi-stage estimator of SAQ requires to store more statistics of the dataset and consume more space for extra factors. However, this overhead is negligible for large-scale datasets.

Code adjustment iteration. We compare the quantization accuracy with different number of code adjustment iterations r in Algorithm 1. We measure accuracy by the average relative error, which is the same as in our main results.

As reported in Figure 10, without any code adjustment ($r = 0$), the average relative error is the worst. As r increases, the average relative errors become significantly better and almost optimal (the error is only 0.7% worse than the optimal when $B = 4, r = 32$). The results show that the first few rounds of iterations can significantly refine the quantization code and produce a good enough error while increasing r to a higher value only slightly improves accuracy. Thus, we recommend setting $r \in [4, 8]$ for CAQ, which can achieve a good balance between accuracy and efficiency.

Memory access for multi-stage estimator. We also quantify the memory access and computational costs of the multi-stage estimator of SAQ. We measure the average bits accessed by the multi-stage estimator when processing an ANNS query, which is the number of bits of quantization

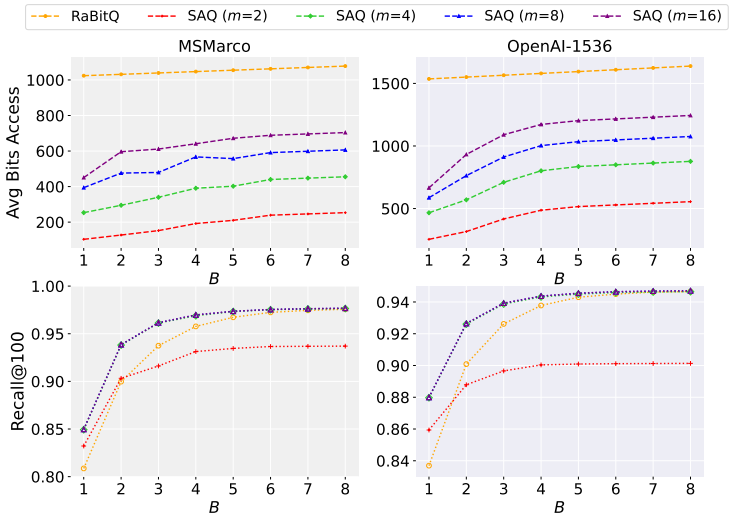


Fig. 11. The average number of bits accessed for each vector (top, smaller the better) and query recall (bottom, $nprob = 200$) for the estimator of RaBitQ and multi-stage estimator of SAQ with different m under different B (the recall curves of $m = \{4, 8, 16\}$ are almost identical).

codes required by the estimator for a single distance estimation. As defined in Section 4.3, the estimator confidence level is governed by the parameter m in $\text{Est}_v(Seg) = \sigma_{Seg} \cdot m$. As m increases, the confidence in the lower bound distance produced by the estimator increases, though it is likely that candidates are pruned. When setting m to a small value, pruning becomes radical and it may weaken the recall.

As reported in Figure 11, the average number of bits accessed by a query is always larger than or equal to the number of dimensions of the dataset. This is because RaBitQ always estimates the distance with the most significant bit of the quantization code. RaBitQ also uses this distance to prune candidates, and so the bits accessed increases slowly as B (i.e., the number of bits for a dimension) increases. For SAQ's multi-stage estimator, the memory access is consistently reduced as m decreases, which means that candidates are pruned earlier before accessing more bits and computing a more accurate distance. We also observe that the multi-stage estimator almost does not harm the recall when $m \geq 4$, but the average of bits accessed increases correspondingly for larger m. When $m = 2$, although the bits accessed decreases dramatically, the recall is also lower by the more radical pruning. Thus, we recommend setting $m = 4$ as default for the multi-stage estimator in SAQ, which can significantly reduce bits accessed, while avoiding weakening recall. SAQ's multi-stage estimator using default m can reduce the bits accessed by about 1.9-4.0× compared with the estimator of RaBitQ.

Accuracy of progressive distance approximation. We compare the relative error produced by the progressive distance (red curve) with the error of the native distance of CAQ (blue curve) and LVQ (green curve). The quantization code for progressive distance is sampled from the native 8-bit quantization code to different b bits.

As reported in Figure 12, the error of the sampled quantization code is consistently lower than that of LVQ and is almost the same as that of the native quantization code when $b > 4$ and the error is slightly larger when $2 \leq b \leq 4$. The small increase in error when reusing the factor under small $b \leq 4$ because the b-bit code is sampled from the 8-bit quantization code and the estimator

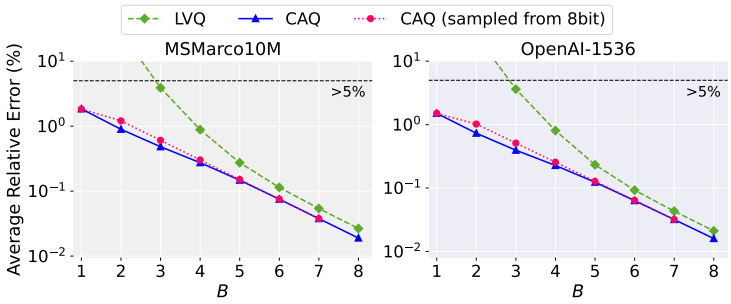


Fig. 12. Progressive distance approximation: b bits sampled from 8-bit CAQ and compared with native b-bit CAQ.

factor is optimized to fit the full 8-bit code, which enlarges the gap between the reuse factor and the native factor when more bits of code are dropped. However, when $b = 1$, the error of the sampled quantization code and the native one is almost the same, since the estimator factor is highly concentrated around 0.8 when $b = 1$ [18] and the estimator does not rely on the factor produced by 8-bit quantization.

6 Related Work

Clustering-based vector quantization. Product quantization (PQ) is a classic method that divides the D-dimensional vector space into M sub-spaces with dimension D/M and uses K-means clustering to obtain K vector codewords for each sub-space [5, 23, 34, 51]. The codewords for each sub-space forms a codebook, and a PQ has M codebooks. Optimized PQ (OPQ) improves PQ by applying a rotation matrix R beforehand to evenly distribute the energy among the M sub-spaces [19]. Locally optimized PQ (LOPQ) works with the IVF index and trains a set of codebooks for each cluster of vectors to better adapt to local data distribution [25]. Instead of partitioning a vector dimension as in PQ, residue quantization (RQ) uses codebooks that are in the same D-dimensional space of the original vectors [5, 23, 34, 51]. The codebooks are trained sequentially with K-means with each codebook working on the residues from all its previous codebooks. To improve accuracy, additive quantization (AQ) improves RQ by jointly learning the M D-dimensional codebooks [5]. However, the complexity is high for both learning the codebooks and encoding each vector with the codebooks. To speed up AQ, LSQ [33] and LSQ++ [34] conduct both algorithm and system optimizations by reformulating the codebook learning problem and using GPU for acceleration. There are also vector quantization methods with other codebook structures, e.g., composite quantization (CQ) [50] and tree quantization (TreeQ) [13].

Clustering based methods use lookup tables (LUTs) for efficient approximate distance computation [5, 23, 34, 51]. A query first computes its distances to all the codewords to initialize the LUTs, and to compute the approximate distance with a vector x, the distances of x's codewords are aggregated over the codebooks. Typically, a codebook size of $K = 256$ is used such that the index of each codeword can be stored as a byte. However, it is observed that table lookup does not utilize CPU registers efficiently with $K = 256$ [3], and thus it is proposed to use smaller codebooks (i.e., $K = 16$) to fit each LUT into the registers [45]. However, at the same space quota for each vector, smaller codebooks degrade accuracy.

Projection-based vector quantization. It is well known that the energy of high-dimension vectors concentrate on the leading dimensions after projection with the PCA matrix. As such,

FEXIPRO [27] adopts the leading dimensions to compute similarity bounds for efficient pruning in maximum inner product search (MIPS). DADE uses PCA-based dimension reduction for Euclidean distance and derives probabilistic bounds for the approximate distance computed with the leading dimensions [9]. A progressive distance computation method is introduced, where a vector first uses its leading dimensions for computation and then checks whether its distance to query is smaller than the current top-k with high probability, and exact distance is only computed if the check passes. In a similar vein, ADSampling projects the vectors with a random orthonormal projection matrix and shows that a unbiased distance estimation can be obtained by sampling some dimensions [9]. LeanVec learns the projection matrix for dimension reduction in a query-aware manner [46], while GleanVec [47] improves LeanVec by learning multiple projection matrices to adapt to local data distributions like LOPQ. Different from the dimension reduction methods, RabitQ uses the random orthonormal projection matrix such that vector dimensions have similar magnitude and quantizes each dimension with 1-bit [18]. To use more bits for each dimension, RabitQ pads the original vector with zeros to increase the dimension before projection. Observing that padding yields inferior accuracy, extended RabitQ (E-RabitQ) uses codewords on the unit sphere after projection and is shown to be asymptotically optimal in accuracy [16]. Currently, E-RabitQ represents the state-of-the-art in accuracy. As we demonstrated in Section 5, SAQ significantly improves E-RabitQ in terms of both encoding speed and quantization accuracy.

7 Conclusions

We have presented SAQ, an advanced vector quantization method that addresses critical limitations of existing methods in both encoding efficiency and quantization accuracy through dimension segmentation and code adjustment. Extensive experiments demonstrate that SAQ significantly outperforms state-of-the-art methods, achieving up to 80% lower quantization error and 80× faster encoding speeds compared to Extended RabitQ. These advancements position SAQ as a robust solution for high-accuracy, efficient vector quantization in large-scale ANNS applications.

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 (July 2023), 3433–3446. doi:10.14778/3611479.3611537

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

[3] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2016. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *42nd International Conference on Very Large Data Bases*, Vol. 9. 12.

[4] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374. doi:10.1016/j.is.2019.02.006

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

[6] Matina Charami, Rami Halloush, and Sofia Tsekeridou. 2007. Performance evaluation of TreeQ and LVQ classifiers for music information retrieval. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 331–338.

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

[8] Cohere. [n. d.]. *Cohere Embed V3*. https://cohere.com/blog/introducing-embed-v3 (2023).

[9] Liwei Deng, Penghao Chen, Ximu Zeng, Tianfu Wang, Yan Zhao, and Kai Zheng. 2024. Efficient Data-Aware Distance Comparison Operations for High-Dimensional Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 18, 3 (Nov. 2024), 812–821. doi:10.14778/3712221.3712244

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[11] Elastic. [n. d.]. *Elastic*. https://www.elastic.co/enterprise-search/vector-search (2024).
[12] Faiss. [n. d.]. *Faiss*. https://github.com/facebookresearch/faiss (2023).
[13] Jonathan T Foote. 2003. TreeQ Manual V0. 8.
[14] 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 (2019), 461–474. doi:10.14778/3303753.3303754
[15] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 541–552. doi:10.1145/2213836.2213898
[16] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2025. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 3 (2025), 202:1–202:26. doi:10.1145/3725413
[17] 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 (2023), 137:1–137:27. doi:10.1145/3589282
[18] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 3, Article 167 (May 2024), 27 pages. doi:10.1145/3654970
[19] 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.
[20] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. 2016. Quantization based fast inner product search. In *Artificial intelligence and statistics*. PMLR, 482–490.
[21] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2553–2561.
[22] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc. VLDB Endow.* 9, 1 (Sept. 2015), 1–12. doi:10.14778/2850469.2850470
[23] 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.
[24] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. doi:10.1109/TPAMI.2010.57
[25] Yannis Kalantidis and Yannis Avrithis. 2014. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2321–2328.
[26] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.
[27] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. FEXIPRO: fast and exact inner product retrieval in recommender systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 835–850.
[28] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2020), 1475–1488. doi:10.1109/TKDE.2019.2909204
[29] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. *Science* 378, 6624 (2022), 1092–1097.
[30] Xianglong Liu, Lei Huang, Cheng Deng, Bo Lang, and Dacheng Tao. 2016. Query-adaptive hash code ranking for large-scale multi-view visual search. *IEEE Transactions on Image Processing* 25, 10 (2016), 4514–4524.
[31] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.
[32] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.
[33] Julieta Martinez, Joris Clement, Holger H Hoos, and James J Little. 2016. Revisiting additive quantization. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*. Springer, 137–153.
[34] Julieta Martinez, Shobhit Zakhmi, Holger H Hoos, and James J Little. 2018. LSQ++: Lower running time and higher recall in multi-codebook quantization. In *Proceedings of the European conference on computer vision (ECCV)*. 491–506.
[35] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. 2018. [Invited Paper] A Survey of Product Quantization. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 2–10. doi:10.3169/mta.6.2

[36] 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. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.
[37] OpenAI. [n. d.]. *New embedding models and API updates.* https://openai.com/index/new-embedding-models-and-api-updates/ (2024).
[38] Marco Patella and Paolo Ciaccia. 2008. The Many Facets of Approximate Similarity Search. In *First International Workshop on Similarity Search and Applications (sisap 2008).* 10–21. doi:10.1109/SISAP.2008.18
[39] pgvector. [n. d.]. *pgvector.* https://github.com/pgvector/pgvector (2024).
[40] Qdrant. [n. d.]. *Qdrant.* https://qdrant.tech/ (2024).
[41] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PmLR, 8748–8763.
[42] Nina Shvetsova, Brian Chen, Andrew Rouditchenko, Samuel Thomas, Brian Kingsbury, Rogerio S Feris, David Harwath, James Glass, and Hilde Kuehne. 2022. Everything at once-multi-modal fusion transformer for video retrieval. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*. 20020–20029.
[43] SingleStore. [n. d.]. *SingleStore.* https://www.singlestore.com/built-in-vector (2024).
[44] Yongye Su, Yinqi Sun, Minjia Zhang, and Jianguo Wang. 2024. Vexless: A Serverless Vector Data Management System Using Cloud Functions. *Proc. ACM Manag. Data* 2, 3, Article 187 (May 2024), 26 pages. doi:10.1145/3654990
[45] Philip Sun, David Simcha, Dave Dopson, Ruiqi Guo, and Sanjiv Kumar. 2023. SOAR: improved indexing for approximate nearest neighbor search. *Advances in Neural Information Processing Systems* 36 (2023), 3189–3204.
[46] Mariano Tepper, Ishwar Singh Bhati, Cecilia Aguerrebere, Mark Hildebrand, and Theodore L. Willke. 2024. LeanVec: Searching vectors faster by making them fit. *Trans. Mach. Learn. Res.* 2024 (2024). https://openreview.net/forum?id=wczqrpOrIc
[47] Mariano Tepper, Ishwar Singh Bhati, Cecilia Aguerrebere, and Ted Willke. 2024. GleanVec: Accelerating vector search with minimalist nonlinear dimensionality reduction. *CoRR* abs/2410.22347 (2024). arXiv:2410.22347 doi:10.48550/ARXIV.2410.22347
[48] TREC-RAG. [n. d.]. *TREC-RAG Corpus 2024.* https://trec-rag.github.io/annoucements/2024-corpus-finalization/ (2024).
[49] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.
[50] Jingdong Wang and Ting Zhang. 2018. Composite quantization. *IEEE transactions on pattern analysis and machine intelligence* 41, 6 (2018), 1308–1322.
[51] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2017. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2017), 769–790.
[52] Stephen J Wright. 2015. Coordinate descent algorithms. *Mathematical programming* 151, 1 (2015), 3–34.
[53] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=zeFrfgyZln
[54] Mingyu Yang, Wentao Li, and Wei Wang. 2025. Fast High-dimensional Approximate Nearest Neighbor Search with Efficient Index Time and Space. arXiv:2411.06158 [cs.DB] https://arxiv.org/abs/2411.06158
[55] 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*. 2241–2253.
[56] Peiqi Yin, Xiao Yan, Qihui Zhou, Hui Li, Xiaolu Li, Lin Zhang, Meiling Wang, Xin Yao, and James Cheng. 2025. Gorgeous: Revisiting the Data Layout for Disk-Resident High-Dimensional Vector Search. *CoRR* abs/2508.15290 (2025). arXiv:2508.15290 doi:10.48550/ARXIV.2508.15290

Received April 2025; revised July 2025; accepted August 2025