RaBitQ

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

Castor Shem

University of Waterloo | October 29, 2026

1 Introduction

Notation

- Vectors are **bolded**: $\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ ... \\ x_{n-1} \end{bmatrix}$
- Inner product denoted by angle brackets:
  - $\langle \mathbf{x}, \mathbf{q} \rangle = \sum_{i=0}^{n-1} x_i q_i = x_0 q_0 + x_1 q_1 + ... + x_{n-1} q_{n-1}$
- Euclidean norm: $||\mathbf{x}|| = \sqrt{x_0^2 + x_1^2 + ... + x_{n-1}^2}$
  - Squared norm: $||\mathbf{x}||^2 = x_0^2 + x_1^2 + ... + x_{n-1}^2$

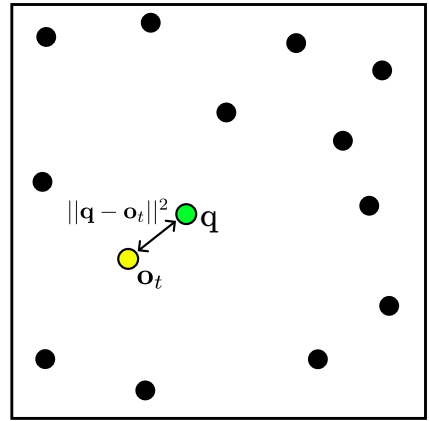
Nearest-Neighbour Search

Given:

- dataset of vectors
  $\mathcal{D} = \{\mathbf{o}_0, \mathbf{o}_1, ..., \mathbf{o}_n\}$
- query vector $\mathbf{q}$

Find $\mathbf{o}_t \in \mathcal{D}$ that is nearest to $\mathbf{q}$

Can be very slow when dimensionality is high



Approximate NNS

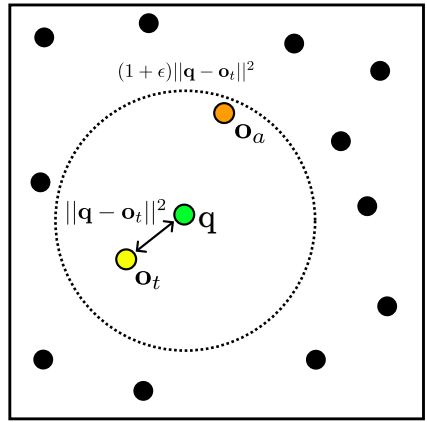
Given:

- dataset of vectors
  $\mathcal{D} = \{\mathbf{o}_0, \mathbf{o}_1, ..., \mathbf{o}_n\}$
- query vector $\mathbf{q}$

Find a *close enough* answer

That is, find a vector $\mathbf{o}_a$ within $(1+\epsilon)$ times the distance to the true nearest neighbour $\mathbf{o}_t$

This can still be very slow



Quantization

- ANNS algorithms require the distance between query and data vectors $||\mathbf{q} - \mathbf{o}_i||^2$
- But, computing distance is expensive
- Solution: create small “quantizations” of data vectors and use them to estimate distance
  - Denote quantization of $\mathbf{o}$ as $\bar{\mathbf{o}}$

One Method: Product Quantization

- One popular method is Product Quantization (PQ)
  - Split each vector into m subvectors (m subspaces)
  - For each subspace, cluster subvectors and obtain k centroids
  - Quantization of a vector is constructed from the centroids corresponding to its subvectors
- PQ is empirically successful on many datasets, but *has no theoretical guarantee* for the error bound of the approximate distance
  - It happens to work well most of the time, but we can't predict when it might fail

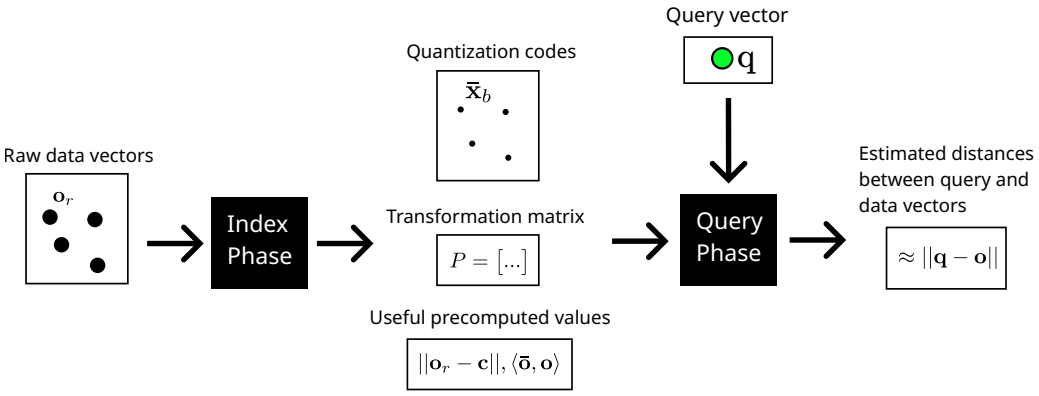
Proposal: RaBitQ

- **Ra**ndomly-Transformed **Bi**-Valued Vec**t**ors
- A quantization method with theoretical guarantees
- More accurate and efficient distance estimations than PQ (and variants)



Image: JM Ligero Loarte via Wikimedia Commons, CC BY 3.0

RaBitQ Algorithm (Bird's-Eye View)



2 Algorithm: Index Phase

Steps

1. Normalize raw data vectors
2. Create codebook
3. Randomly transform codebook
4. Match data vectors to codebook vectors
5. Store small quantization codes

Normalizing Raw Data Vectors

Let $\mathbf{c}$ be centroid of raw data vectors. The normalization of data vector $\mathbf{o}_r$ is:

$$\mathbf{o} = \frac{\mathbf{o}_r - \mathbf{c}}{||\mathbf{o}_r - \mathbf{c}||}$$

Things to observe:

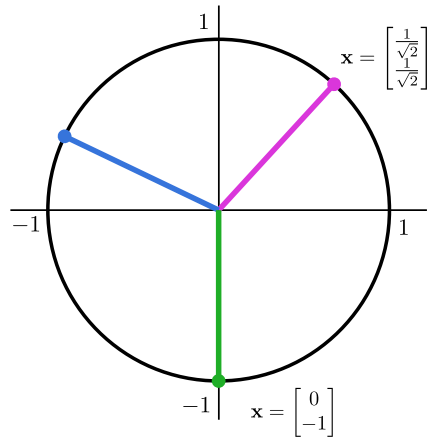
- Vectors spread out (somewhat) evenly about origin, since we subtracted $\mathbf{c}$
- Vectors now all unit vectors

The dataset vectors are now spread out (somewhat) evenly on the D-dimensional unit hypersphere (where D is dimensionality of dataset vectors).

Aside: Unit Hypersphere

D-dimensional unit hypersphere: space of all D-dimensional unit vectors.

The 2-dimensional unit hypersphere is the unit circle $\{\mathbf{x} : ||\mathbf{x}|| = 1\}$



Creating the Codebook: Initial Approach

- Data vectors are spread out (somewhat) evenly on the D-dimensional unit hypersphere.
- $\Rightarrow$ The codebook should also be spread out evenly on the unit hypersphere.

$$\mathcal{C} = \left\{ \frac{1}{\sqrt{D}}, \frac{-1}{\sqrt{D}} \right\}^D$$

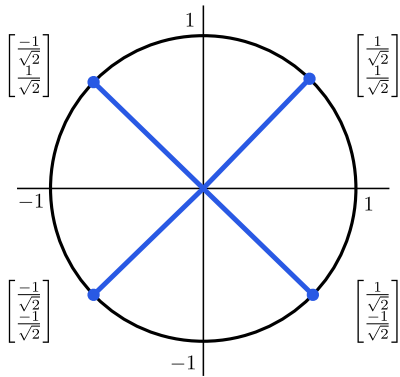
Variables

D: Dimensionality of data vectors

Creating the Codebook: 2D Example

For $D = 2$, the codebook $\mathcal{C}$ would be:

$$\mathcal{C} = \left\{ \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}} \right\}^2$$



Codebook Transformation

A random orthogonal matrix P is generated and applied to the codebook vectors (which rotates them randomly).

$$\mathcal{C}_{rand} = \{P\mathbf{x} \mid \mathbf{x} \in \mathcal{C}\}$$

Intuition: applying a random rotation prevents the codebook from favouring particular vectors.

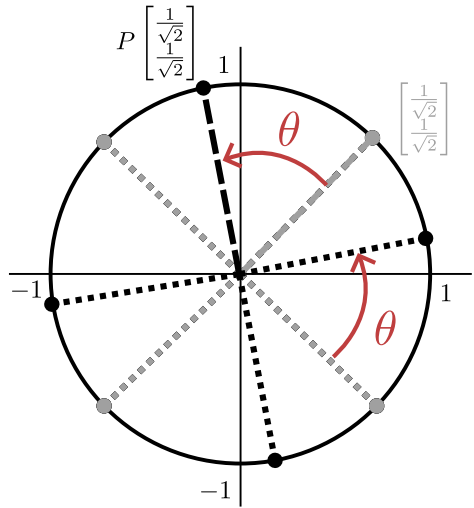
Codebook Transformation (Example)

For example, in $D = 2$:
Choose a random orthogonal matrix.

$$P = \begin{bmatrix} 0.500 & -0.866 \\ 0.866 & 0.500 \end{bmatrix}$$

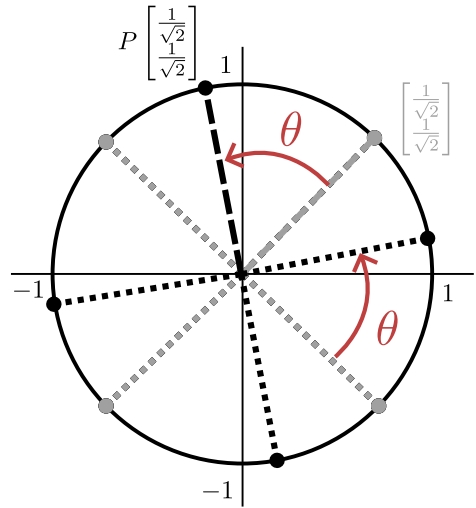
For each vector in codebook, apply transformation.

$$\mathcal{C}_{rand} = \{P\mathbf{x} \mid \mathbf{x} \in \mathcal{C}\}$$



Codebook Transformation (Example)

$\mathbf{x}$	$P\mathbf{x}$
$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} -0.259 \\ 0.966 \end{bmatrix}$
$\begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} 0.966 \\ 0.259 \end{bmatrix}$
$\begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} -0.966 \\ -0.259 \end{bmatrix}$
$\begin{bmatrix} -1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} 0.259 \\ -0.966 \end{bmatrix}$



Quantization Codes

Quantization codes $\mathbf{x}_b$ are bit strings that correspond to the signs of the codebook vectors.

$$\mathbf{x} = \frac{2\mathbf{x}_b - \mathbf{1}_D}{\sqrt{D}} \qquad \mathbf{x}_b = \frac{\sqrt{D}\mathbf{x} + \mathbf{1}_D}{2}$$

$\mathbf{x}$	$P\mathbf{x}$	$\mathbf{x}_b$
$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} -0.259 \\ 0.966 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$
$\begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} 0.966 \\ 0.259 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
$\begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} -0.966 \\ -0.259 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$
$\begin{bmatrix} -1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} 0.259 \\ -0.966 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Match Data Vectors to Codebook Vectors

The quantization of (normalized) data vector $\mathbf{o}$ is its nearest transformed codebook vector $P\bar{\mathbf{x}} \in \mathcal{C}_{rand}$.

Nearest Vector $\Leftrightarrow$ Max Inner Product

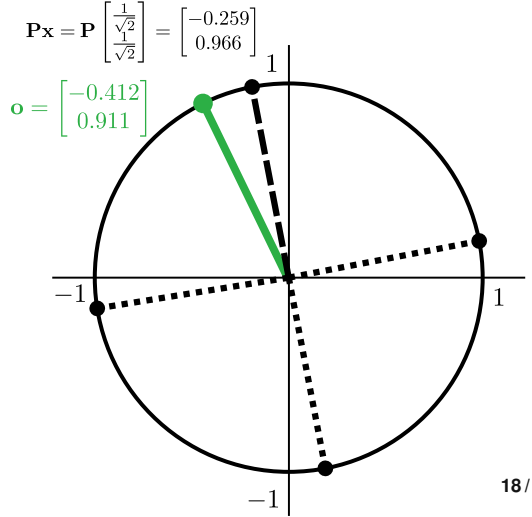
If $\mathbf{o}$ and $\mathbf{x} \in \mathcal{C}$ are unit vectors, the nearest codebook vector $\bar{\mathbf{x}}$ to $\mathbf{o}$ is...

$$\bar{\mathbf{x}} = \arg\min_{\mathbf{x}\in\mathcal{C}} ||\mathbf{o} - P\mathbf{x}||^2 = \arg\max_{\mathbf{x}\in\mathcal{C}} \langle \mathbf{o}, P\mathbf{x} \rangle$$

Match Data Vectors to Codebook Vectors (Example)

Here, $\mathbf{o} = \begin{bmatrix} -0.412 \\ 0.911 \end{bmatrix}$ is mapped to $P\bar{\mathbf{x}} = \begin{bmatrix} -0.259 \\ 0.966 \end{bmatrix}$ and gets the quantization code 11.

$\mathbf{x}$	$P\mathbf{x}$	$\mathbf{x}_b$
$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} -0.259 \\ 0.966 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$



Inverse Transformation

We identify the nearest codebook vector by solving $\arg\max_{\mathbf{x}\in\mathcal{C}} \langle \mathbf{o}, P\mathbf{x} \rangle$.

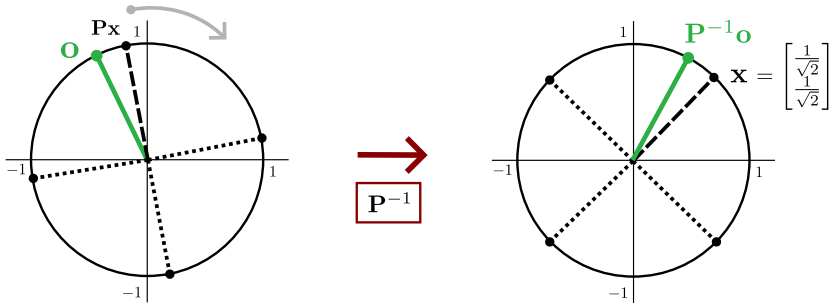
Problem: applying P to every codebook vector $\mathbf{x} \in \mathcal{C}$ would be very expensive.

$$\mathcal{C} = \left\{ \frac{1}{\sqrt{D}}, \frac{-1}{\sqrt{D}} \right\}^D \longrightarrow |\mathcal{C}| = 2^D$$

For $D = 100$, we would be transforming $2^D = 2^{100}$ vectors.

Inverse Transformation

Solution: instead of transforming codebook vectors, transform the data vectors.



$$\arg\max_{\mathbf{x}\in\mathcal{C}} \langle \mathbf{o}, P\mathbf{x} \rangle = \arg\max_{\mathbf{x}\in\mathcal{C}} \left\langle P^{-1}\mathbf{o}, \mathbf{x} \right\rangle$$

Inverse Transformation (Example)

To obtain the quantization $\bar{\mathbf{o}} = P\bar{\mathbf{x}}$ for a data vector $\mathbf{o}$, we only need to apply a transformation to $\mathbf{o}$ and look at the signs of its elements.

$$\mathbf{o} = \begin{bmatrix} -0.412 \\ 0.911 \end{bmatrix}$$

$$P^{-1}\mathbf{o} = \begin{bmatrix} 0.500 & -0.866 \\ 0.866 & 0.500 \end{bmatrix} \begin{bmatrix} -0.412 \\ 0.911 \end{bmatrix} = \begin{bmatrix} 0.929 \\ 0.369 \end{bmatrix}$$

$$\Rightarrow \qquad \bar{\mathbf{x}} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \qquad \bar{\mathbf{x}}_b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$\mathbf{x}$	$\mathbf{x}_b$
$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$
$\begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
...	...

3. Quantization Codes

Now, for data vector $\mathbf{o}$ we can obtain its quantization $\bar{\mathbf{o}} = P\bar{\mathbf{x}}$ and a small binary string $\bar{\mathbf{x}}_b$ that can be used to reconstruct it.

$$\bar{\mathbf{x}}_b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\bar{\mathbf{x}} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

$$\bar{\mathbf{o}} = P\bar{\mathbf{x}} = \begin{bmatrix} -0.259 \\ 0.966 \end{bmatrix}$$

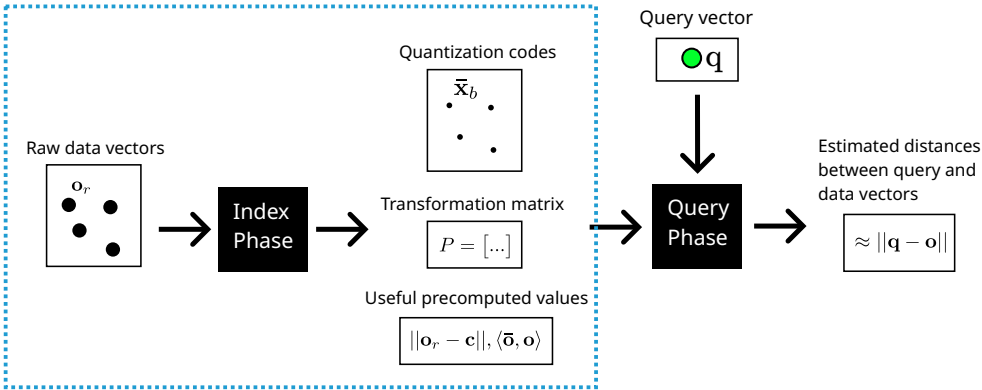
Variables

- $\mathbf{o}$: Data vector
- $\bar{\mathbf{o}}$: Quantization of $\mathbf{o}$
- $\bar{\mathbf{x}}$: Codebook vec. nearest $P^{-1}\mathbf{o}$
- $\bar{\mathbf{x}}_b$: Quantization code of $\mathbf{o}$
- P : Transformation matrix

Summary: Steps

1. Normalize raw data vectors
2. Create codebook
3. Randomly transform codebook
4. Match data vectors to codebook vectors
5. Store small quantization codes

Summary: Diagram



3 Algorithm: Query Phase

Steps

1. Normalize raw query vector
2. Obtain estimator between query and data vectors
3. Efficiently compute estimator

Normalize Query Vector

Normalize query vector as we did for the data vectors in the index phase.

$$\mathbf{q} = \frac{\mathbf{q}_r - \mathbf{c}}{||\mathbf{q}_r - \mathbf{c}||}$$

Variables

$\mathbf{q_r}$: Raw query vector

$\mathbf{q}$: Normalized query vector

$\mathbf{c}$: Centroid of data vectors

Compute Estimator

We want a term that estimates the distance between $\mathbf{o}_r$ and $\mathbf{q}_r$.

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

Compute Estimator

We want a term that estimates the distance between $\mathbf{o}_r$ and $\mathbf{q}_r$.

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

- $||\mathbf{o}_r - \mathbf{c}||^2$ can be precomputed during index phase
- $||\mathbf{q}_r - \mathbf{c}||^2$ can be computed once during query phase and reused for all data vectors
- The problem of estimating $||\mathbf{o}_r - \mathbf{q}_r||^2$ reduces to the problem of estimating $\langle \mathbf{o}, \mathbf{q} \rangle$.

Deriving the Estimator

Lemma 3.1

Let $\mathbf{o}, \mathbf{q}, \bar{\mathbf{o}}$ be any three unit vectors. When $\mathbf{o}, \mathbf{q}$ are colinear (i.e. $\mathbf{o} = \pm\mathbf{q}$):

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

When $\mathbf{o} \neq \pm\mathbf{q}$:

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

where $\mathbf{e}_1 = \frac{\mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o}}{||\mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o}||}$.

Deriving the Estimator

Lemma 3.1

Let $\mathbf{o}, \mathbf{q}, \bar{\mathbf{o}}$ be any three unit vectors.

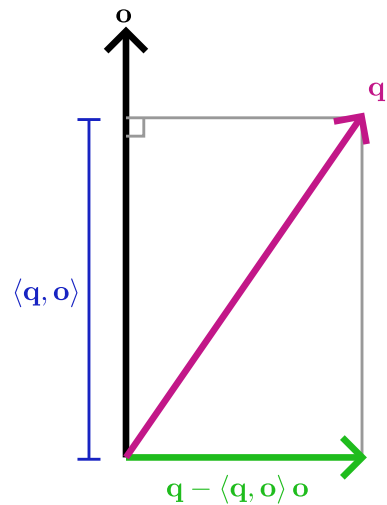
When $\mathbf{o}, \mathbf{q}$ are colinear (i.e. $\mathbf{o} = \pm\mathbf{q}$):

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

When $\mathbf{o} \neq \pm\mathbf{q}$:

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

where $\mathbf{e}_1 = \frac{\mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o}}{\| \mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o} \|}$.



Lemma 3.1: Colinear Case

Lemma 3.1

When $\mathbf{o}, \mathbf{q}$ are colinear: $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{o}, \mathbf{q} \rangle$ ...

In the colinear case, we can easily obtain $\langle \mathbf{o}, \mathbf{q} \rangle$:

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

Lemma 3.1: Colinear Case

Lemma 3.1

When $\mathbf{o}, \mathbf{q}$ are colinear: $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{o}, \mathbf{q} \rangle$ ...

In the colinear case, we can easily obtain $\langle \mathbf{o}, \mathbf{q} \rangle$:

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

- $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ can be precomputed during index phase
- $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ can be **efficiently computed** during query phase

Lemma 3.1: Non-Colinear Case

Lemma 3.1

... When $\mathbf{o}, \mathbf{q}$ are not colinear: $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{o}, \mathbf{q} \rangle + \langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle \cdot \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2}$ where $\mathbf{e}_1 = \frac{\mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o}}{||\mathbf{q} - \langle \mathbf{q}, \mathbf{o} \rangle \mathbf{o}||}$.

In the non-colinear case, we cannot compute $\langle \mathbf{o}, \mathbf{q} \rangle$ easily because of $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$.
But maybe we can use $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ as a good estimate?

Lemma 3.1: Non-Colinear Case

Rewrite $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ using Lemma 3.1...

$$\begin{aligned}\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} &= \frac{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{o}, \mathbf{q} \rangle + \langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle \cdot \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2}}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} \\ &= \langle \mathbf{o}, \mathbf{q} \rangle + \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2} \cdot \frac{\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}\end{aligned}$$

Intuition: This expression is just like our exact expression $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} = \langle \mathbf{o}, \mathbf{q} \rangle$ from the colinear case, but with an error term

Analysis of Distributions

Result of Analysis

For $D \in [10^2, 10^6]$,

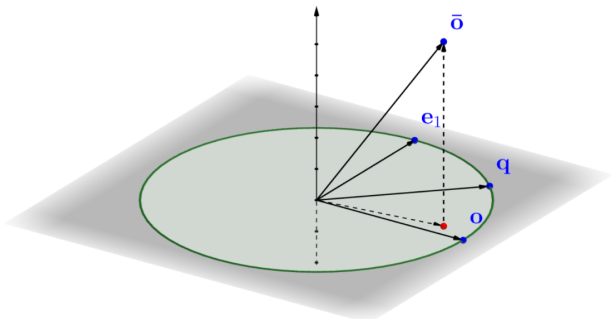
$$\mathbb{E}[\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle] \approx 0 \qquad \text{and} \qquad \mathbb{E}[\langle \bar{\mathbf{o}}, \mathbf{o} \rangle] \approx 0.8$$

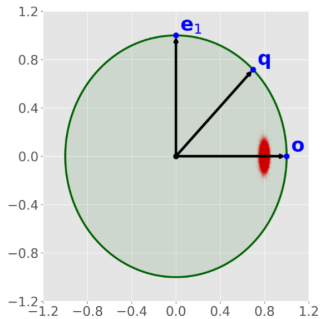
Detailed proof in Appendix B of the paper, based on analysis of the distribution of the random variables.

Empirical Verification of Analysis

$\mathbb{E}[\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle] \approx 0$ and $\mathbb{E}[\langle \bar{\mathbf{o}}, \mathbf{o} \rangle] \approx 0.8$

1. For fixed $\mathbf{o}, \mathbf{q}$, sample 10^5 random P (producing 10^5 random $\bar{\mathbf{o}}$)
2. Compute $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$





Unbiasedness of Estimator

Theorem 3.2

$\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ is an unbiased estimate of $\langle \mathbf{o}, \mathbf{q} \rangle$.

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

The error bound of the estimate is:

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

Detailed proof in Appendix C of the paper.

3.5 Algorithm: Efficiently Computing the Estimator

Efficiently Approximating the Estimator

From Lemma 3.1, our estimator is:

$$\langle \mathbf{o}, \mathbf{q} \rangle \approx \frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$$

To compute the estimator:

- $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ can be precomputed during index phase
- $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ can be **efficiently computed** with low error during query phase

Efficiently Computing the Estimator

As before, instead of transforming $\bar{\mathbf{o}}$, transform $\mathbf{q}$.

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

Scalar Quantization

We will first use uniform scalar quantization to quantize $\mathbf{q}' = P^{-1}\mathbf{q}$ into a vector of B_q-bit unsigned integers.

$$\mathbf{q}' = \begin{bmatrix} 0.32 & -0.51 & -0.06 & 0.45 & 0.13 & 0.64 \end{bmatrix}$$

Scalar Quantization

Identify minimum (v_l) and maximum (v_r) values in $\mathbf{q}'$.

$$\mathbf{q}' = \begin{bmatrix} 0.32 & -0.51 & -0.06 & 0.45 & 0.13 & 0.64 \end{bmatrix}$$

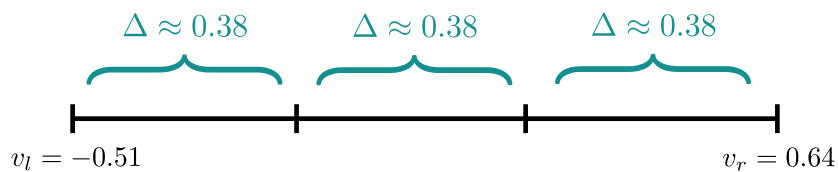
$$v_l = -0.51$$

$$v_r = 0.64$$

Scalar Quantization

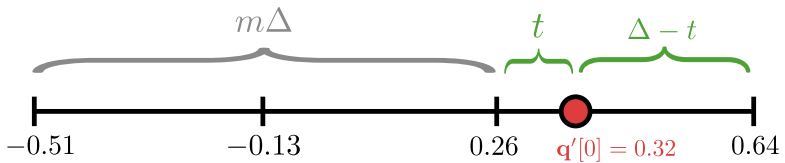
Split range of values $[v_l, v_r]$ into 2^{B_q-1} segments of equal length Δ. For $B_q = 2$...

$$\Delta = \frac{v_r - v_l}{2^{B_q} - 1} = \frac{0.64 - (-0.51)}{2^2 - 1} \approx 0.38$$



Scalar Quantization (Example)

Round each element to one of its boundaries...

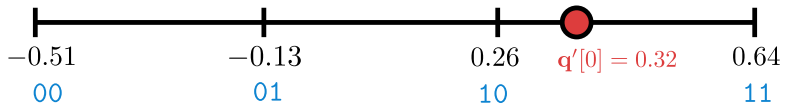


$$\mathbf{q}'[0] = 0.32 = v_l + m\Delta + t \Rightarrow m = 2, t = 0.06$$

$$\bar{\mathbf{q}}[0] = \begin{cases} 0.26 \text{ with probability } \frac{t}{\Delta} = 0.17 \\ 0.64 \text{ with probability } \frac{\Delta - t}{\Delta} = 1 - \frac{t}{\Delta} = 0.83 \end{cases}$$

Scalar Quantization (Example)

And assign it the corresponding B_q-bit bitstring.



$$\bar{\mathbf{q}}[0] = \begin{cases} 0.26 \text{ with probability } \frac{t}{\Delta} = 0.17 = \texttt{10} \\ 0.64 \text{ with probability } \frac{\Delta - t}{\Delta} = 1 - \frac{t}{\Delta} = 0.83 = \texttt{11} \end{cases}$$

Scalar Quantization (Example)

$\mathbf{q}'$ represented with $\bar{\mathbf{q}}_u$, a vector of B_q-bit unsigned integers.

m	0	1	2	3
Quantized Value $v_l + m\Delta$	-0.51	-0.13	0.26	0.64
2-Bit Representation	00	01	10	11

$$\mathbf{q}' = \begin{bmatrix} 0.32 & -0.51 & -0.06 & 0.45 & 0.13 & 0.64 \end{bmatrix}$$

$$\bar{\mathbf{q}} = \begin{bmatrix} 0.26 & -0.51 & -0.13 & 0.26 & 0.26 & 0.64 \end{bmatrix}$$

$$\bar{\mathbf{q}}_u = \begin{bmatrix} 10 & 00 & 01 & 10 & 01 & 11 \end{bmatrix}$$

Using the Quantization for Efficient Computation

Our goal was to compute $\langle \bar{\mathbf{x}}, \mathbf{q}' \rangle$. Instead, we can use $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$. It can be shown that the error introduced here is much less than overall error of estimator.

Rewrite $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$:

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

- $\bar{\mathbf{x}}_b$ is the small bit representation of the codebook vector $\bar{\mathbf{x}}$ (which is the codebook vector matching the data vector $\mathbf{o}$).
- $\bar{\mathbf{q}}_u$ is the small bit representation of the query vector $\bar{\mathbf{q}}$, produced by scalar quantization.

Using the Quantization for Efficient Computation

Our goal was to compute $\langle \bar{\mathbf{x}}, \mathbf{q}' \rangle$. Instead, we can use $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$. It can be shown that the error introduced here is much less than overall error of estimator.

Rewrite $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$:

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

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

Using the Bit Representations for Efficient Computation

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

- 2nd term can be precomputed during index phase
- 3rd term can be computed once during query phase
- Problem of computing $\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle$ reduces to problem of computing 1st term, $\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$

Bit Representations for Efficient Computation (Ex.)

$\mathbf{o}$ is a 7D normalized dataset vector, and after *the index phase* its quantization is $P\bar{\mathbf{x}} \in \mathcal{C}_{rand}$.

Its corresponding 7D codebook vector is $\bar{\mathbf{x}}$.

$$\bar{\mathbf{x}} = \begin{bmatrix} \frac{1}{\sqrt{7}} & \frac{1}{\sqrt{7}} & -\frac{1}{\sqrt{7}} & \frac{1}{\sqrt{7}} & -\frac{1}{\sqrt{7}} & -\frac{1}{\sqrt{7}} & \frac{1}{\sqrt{7}} \end{bmatrix}$$

The small bit representation of $\bar{\mathbf{x}}$ is $\bar{\mathbf{x}}_b$ (from the signs of $\bar{\mathbf{x}}$).

$$\bar{\mathbf{x}}_b = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Bit Representations for Efficient Computation (Ex.)

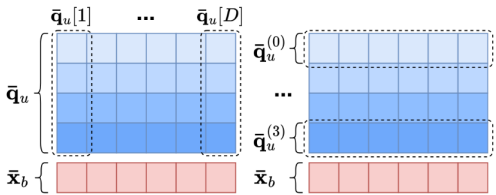
$\mathbf{q}$ is a 7D query vector. After *scalar quantization* with $B_q = 2$, its quantized form is $\bar{\mathbf{q}}$, and the small bit representation of $\bar{\mathbf{q}}$ is:

$$\bar{\mathbf{q}}_u = \begin{bmatrix} 00 & 01 & 10 & 01 & 00 & 11 & 00 \end{bmatrix}$$

Inner Product of Small Bit Representations

$\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$ equals the weighted sum of inner products between $\bar{\mathbf{x}}_b$ and $\bar{\mathbf{q}}_u^{(j)}$

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



- $\bar{\mathbf{x}}_b$ is vector with 1-bit elements
- $\bar{\mathbf{q}}_u$ is vector with B_q-bit elements
- $\bar{\mathbf{q}}_u^{(j)}$ is vector with 1-bit elements (at the jth position of $\bar{\mathbf{q}}_u$)

Inner Product of Small Bit Representations (Example)

Decompose $\bar{\mathbf{q}}_u$ into binary vectors for each of its bit positions:

$$\bar{\mathbf{q}}_u = \begin{bmatrix} 00 & 01 & 10 & 01 & 00 & 11 & 00 \end{bmatrix}$$

$$\bar{\mathbf{q}}_u^{(1)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad \bar{\mathbf{q}}_u^{(0)} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Inner Product of Small Bit Representations (Example)

Fast computation as weighted sum:

$$\langle \bar{\mathbf{x}}, \bar{\mathbf{q}} \rangle = 2^1 \cdot \left\langle \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right\rangle + 2^0 \cdot \left\langle \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right\rangle = 2 \cdot 2 + 1 \cdot 1 = 5$$

This can be computed using (fast) bitwise-and and bitcount operations.

Summary of Derivations in Query Phase

- We want to approximate the distance $||\mathbf{o}_r - \mathbf{q_r}||^2$.
- Earlier, showed the problem of computing $||\mathbf{o}_r - \mathbf{q_r}||^2$ reduces to computing $\langle \mathbf{o}, \mathbf{q} \rangle$.
- $\langle \mathbf{o}, \mathbf{q} \rangle$ can be approximated with the estimator $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ (Theorem 3.2).
  - The denominator can be precomputed, so the problem reduces to computing $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$.
- Earlier, showed that the problem of computing $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ reduces to computing $\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$.

Now, we have a way to efficiently compute $\langle \bar{\mathbf{x}}_b, \bar{\mathbf{q}}_u \rangle$, therefore we can efficiently estimate $||\mathbf{o}_r - \mathbf{q_r}||^2$.

04 Evaluation

Evaluation Information

Two major parts to the evaluation:

1. Quantization algorithm
    - Time and relative error of distance estimation
    - Time taken by index phase
2. ANN search effectiveness
    - QPS, recall, and average distance ratio for ANN search

Evaluation Information

Quantization methods evaluated:

- **PQ:** Popular baseline
- **OPQ:** Variant of PQ
- **LSQ:** Very high accuracy, but very slow index phase
- **RaBitQ**

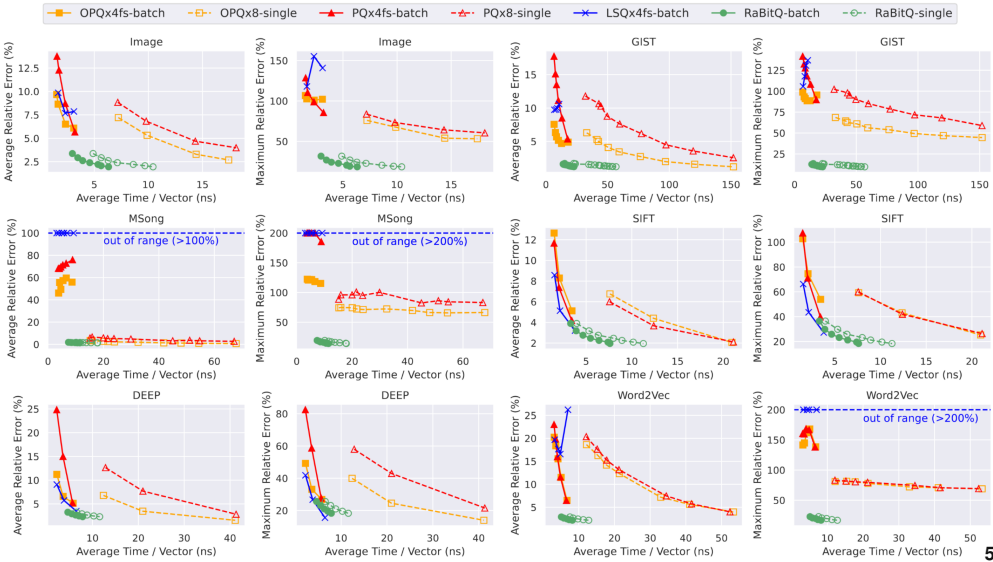
Datasets evaluated (on):

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

Time/Accuracy Tradeoff for Distance Estimation

For each quantization method, estimate distance between a data vector and query vector.

- **Average Relative Error**: Average error between estimated and true distance
- **Maximum Relative Error**: Maximum error between estimated and true distance
- **Time per Vector**: Time taken to produce estimate


OPQx4fs-batch
OPQx8-single
PQx4fs-batch
PQx8-single
LSQx4fs-batch
RaBitQ-batch
RaBitQ-single
Image
Image
GIST
GIST
MSong
MSong
SIFT
SIFT
DEEP
DEEP
Word2Vec
Word2Vec
out of range (>100%)
out of range (>200%)
out of range (>200%)
Average Relative Error (%)
Maximum Relative Error (%)
Average Time / Vector (ns)

Time Cost of Index Phase

For each quantization method, perform its indexing phase. Only the GIST dataset was used.

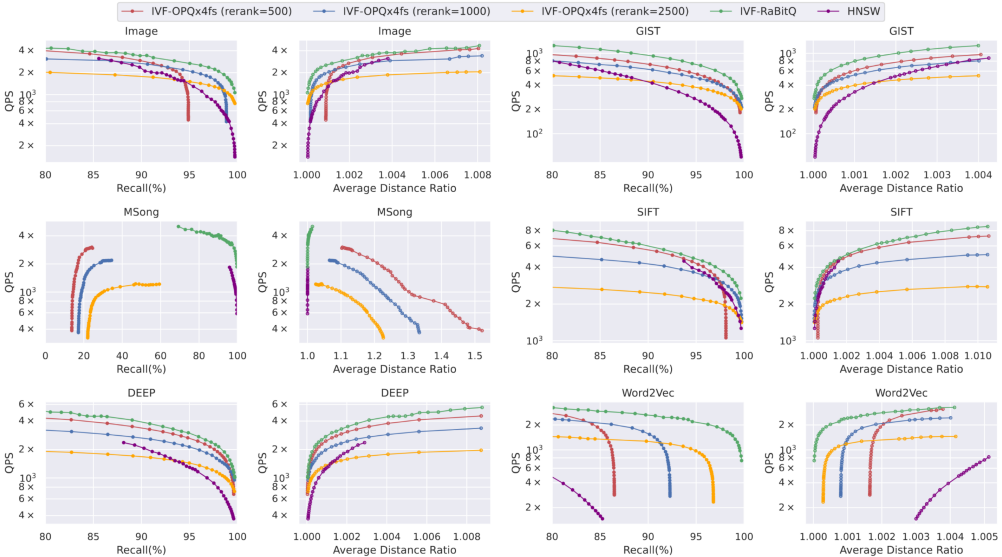
	RaBitQ	PQ	OPQ	LSQ
Time	117s	105s	291s	time-out (>24 hours)

Time/Accuracy Tradeoff for ANN Search

For each quantization method, combine it with the IVF index and perform K-ANN search.

- **Queries per Second (QPS)**: Number of queries a method can handle per second
- **Recall**: Number of retrieved nearest neighbours that are actually true nearest neighbours, divided by K
- **Average Distance Ratio**: Distance ratio of returned data vectors compared to true nearest neighbours, averaged over all K returned data vectors

HNSW is also evaluated as a non-quantization approach.


IVF-OPQx4fs (rerank=500)
IVF-OPQx4fs (rerank=1000)
IVF-OPQx4fs (rerank=2500)
IVF-RaBitQ
HNSW
Image
Image
GIST
GIST
MSong
MSong
SIFT
SIFT
DEEP
DEEP
Word2Vec
Word2Vec
QPS
Recall(%)
Average Distance Ratio

Conclusion

- RaBitQ is a quantization method with a theoretically-guaranteed error bound for its distance approximation
  - **Index phase:** Use random transformation matrix to produce codebook, quantize data vectors
  - **Query phase:** Efficiently compute an unbiased estimator for distance between query and data vectors
- Competitive performance for accuracy, recall, and QPS



Image: JM Ligero Loarte via Wikimedia Commons, CC BY 3.0