Distance Comparison Operation Optimization in ANNS: A Survey and Experimental Evaluation

Bohai Wang
East China Normal University
Shanghai, China
bohaiwang@stu.ecnu.edu.cn

Yanhao Wang
East China Normal University
Shanghai, China
yhwang@dase.ecnu.edu.cn

Huiqi Hu
East China Normal University
Shanghai, China
hqhu@dase.ecnu.edu.cn

Weichen Zhao
East China Normal University
Shanghai, China
weichenzhao@stu.ecnu.edu.cn

Minghao Zhao*
East China Normal University
Shanghai, China
mhzhao@dase.ecnu.edu.cn

Abstract

Approximate nearest neighbor search in high-dimensional spaces is fundamental to vector databases and has numerous practical applications. Distance Comparison Operations (DCOs) have been identified as a major performance bottleneck in approximate nearest neighbor search, prompting the development of various optimization techniques. However, systematic reviews and performance evaluations of these methods remain limited, making it difficult for researchers and practitioners to understand their mechanisms and to select appropriate techniques in specific scenarios. In this paper, we survey and benchmark mainstream DCO optimization methods. We categorize existing approaches into projection pruning- and quantization-based techniques and review representative methods in each category. Extensive experiments across eight datasets, spanning diverse index structures and implementations, evaluate key metrics such as efficiency-accuracy trade-offs and preprocessing overhead of these methods while examining their scalability with respect to data dimensionality and scale. Based on our findings, we provide performance characterizations to guide method selection and highlight promising directions for future research.

Keywords

Approximate nearest neighbor search, Distance comparison optimization, Performance evaluation

1 Introduction

Approximate Nearest Neighbor Search (ANNS) aims to identify the top-k vectors most similar (*i.e.,* nearest) to a given query vector. It covers a wide range of applications in many areas, including vector database queries [35, 36, 39, 42, 54], information retrieval [18, 21, 32, 56], data mining [11, 12, 45], recommender systems [30, 37, 43], and large language models (LLMs) [3, 8, 33]. Compared with exact K-NN search, ANNS effectively mitigates the curse of dimensionality and improves usability in high-dimensional spaces. Until now, numerous ANNS schemes have been proposed in the literature [29].

Despite the wide variety of ANNS methods, they typically follow a similar two-stage paradigm, *i.e.,* first *generating* a set of candidates and then *refining* the candidate set to obtain the query results. Typically, schemes differ in how they generate

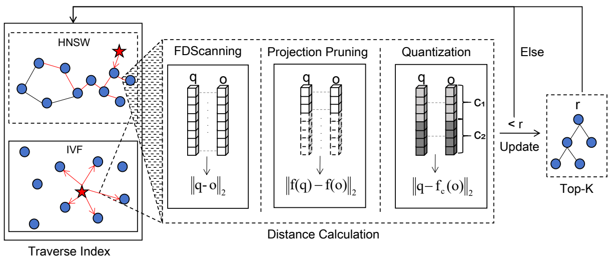


Figure 1: Illustration of DCOs in the ANNS procedure.

candidate vectors but share similar approaches to refining them. To retrieve the K nearest neighbors of a query vector q, a scheme iteratively traverses the index and evaluates the distance between each candidate vector o and the query vector q, *i.e.,* $\mathrm{dist}(o, q)$. A max-heap Q is adopted to maintain the K visited data objects that are closest to the query q. In each round of the iteration, the max-heap Q is updated via *Distance Comparison Operation* (DCO) [13] – once the distance between o and q (*i.e.,* $\mathrm{dist}(o, q) < d_{\max}$) is smaller, the farthest neighbor is removed from Q and o is inserted into Q; otherwise, o is discarded and the next candidate is considered, as illustrated in Figure 1.

Existing studies have shown that the computational cost of DCOs is the dominant component of the total cost in ANNS [13]. As an example, experiments on the 256-dimensional DEEP dataset [6] revealed that DCOs consumed 77.2% of the total execution time in the Hierarchical Navigable Small World (HNSW) graph-based ANNS algorithm [34], and 85% in the Inverted File (IVF)-based ANNS algorithm [23], highlighting DCOs as a primary performance bottleneck in ANNS. In their basic form, DCO techniques rely on full-dimensional distance computations (*e.g., FD-Scanning*), which are computationally expensive. Subsequently, a wide range of *DCO Optimization* methods have been introduced to accelerate ANNS by avoiding full-dimensional vector distance computations. Typical approaches include *Projection Pruning*, which computes distances using only a subset of dimensions, and *Quantization*, which approximates high-dimensional vectors using a few representatives. These optimization methods significantly improve the performance of DCOs.

Unfortunately, unlike index structure optimizations, the effectiveness of DCO techniques is highly sensitive to practical factors such as dimensionality, data scale, query workload, hardware characteristics, and preprocessing overhead. To the best of our knowledge, although several experimental and benchmarking studies have investigated graph-based ANNS [4, 46, 48], they primarily focus on macroscopic, end-to-end performance metrics

*Corresponding author.

EDBT '26, Tampere (Finland)

(*e.g.,* QPS–Recall trade-offs) to assess index structures and search algorithms under specific workloads or hardware configurations. Consequently, the lack of fine-grained, cross-method evaluations that systematically analyze DCO optimizations, disentangle their performance trade-offs, and identify the key factors governing their effectiveness in practice makes it difficult for system designers to make informed decisions when deploying DCO techniques.

Nevertheless, conducting fair comparisons across DCO optimization methods is non-trivial. First, many projection pruning-based methods misleadingly use the ratio of pruned dimensions as an indicator of speedup, implicitly assuming that skipping dimension calculations directly translates into computational acceleration. They often overlook critical factors in real-world systems, such as memory access patterns and branch prediction errors, which can negate the gains of dimension pruning. Second, existing performance evaluation methods are largely detached from modern computing environments. Most have not been assessed under parallel computing models, such as Single Instruction Multiple Data (SIMD), where algorithm performance can vary drastically between scalar and vectorized modes, limiting the applicability of their conclusions. Finally, existing evaluations only consider the trade-off between query efficiency (*i.e.,* Queries Per Second, QPS) and accuracy (*i.e.,* Recall@K), and lack an assessment of fine-grained metrics, *e.g.,* pruning accuracy and preprocessing and memory overhead.

Our Contributions. To address these issues, we provide a systematic review and analysis of DCO optimization methods. We first collect existing approaches and classify them according to their underlying principles for accelerating ANNS. Next, we develop a unified benchmarking framework for DCO optimizations. Using this framework, we conduct an extensive experimental evaluation of 12 representative DCO optimization methods and their variants, covering eight standard datasets, two primary index structures (IVF and HNSW), and two computational modes (scalar and SIMD), accompanied by an in-depth analysis. Our main contributions are summarized as follows:

- We propose a taxonomy of DCO optimization methods based on the principles for accelerating distance computation. Specifically, we categorize them into two types, namely those based on projection pruning [9, 10, 13, 26, 51] and quantization [2, 5, 14, 15, 23, 31]. The taxonomy provides key insights into the DCO optimization landscape and offers a new perspective on the broader field of ANNS methods, in which most existing optimization techniques focus solely on quantization. (§3)
- We conduct a comprehensive performance evaluation of representative DCO optimization methods across eight datasets of multiple modalities, including audio, images, and text. Each method is assessed across several aspects, including query efficiency, accuracy, and preprocessing overhead for index construction and candidate generation. We also examine key factors that affect the effectiveness of optimizations, *e.g.,* dimensionality, data scale, and index configurations. (§4)
- Based on our experimental findings, we provide empirical advice and guidance on selecting the most appropriate DCO optimization in different application scenarios. Furthermore, we analyze key challenges and outline promising directions for future work on this problem. (§5)

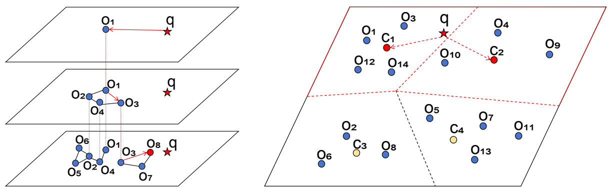
2 Background

Approximate Nearest Neighbor Search: Let O denote a vector database that contains N objects in a D-dimensional Euclidean space $\mathbb{R}^D$. For a query vector $q \in \mathbb{R}^D$, the KNN problem finds the K data objects in O that are closest to the query q. Here, the squared Euclidean distance is used as the distance metric for a query q and any object o, defined as

$$\text{dist}(q, o) = \|q - o\|^2 = \sum_{i=1}^{D} (q_i - o_i)^2 . \quad (1)$$

Obviously, it takes $O(D)$ time to perform each distance computation and $O(ND)$ time to perform a brute-force KNN search. The Approximate KNN Search (ANNS) returns objects that are close to the query q but not necessarily the K closest, thereby significantly improving search efficiency at the expense of accuracy.

Index Structures for ANNS: Index structures are used in ANNS to filter out candidate objects that are irrelevant to the query q, thereby avoiding a brute-force search on the database O in $O(ND)$ time. In this paper, we focus on the two most widely used index structures for ANNS, namely the Hierarchical Navigable Small World (HNSW) [34] and the Inverted File (IVF) [23]. Figure 2 illustrates the schemes of the two indices.



(a) HNSW **(b) IVF**

Figure 2: Illustration of the structures and ANNS procedures of HNSW and IVF.

HNSW [34] is a graph-based ANNS algorithm that features a multilayer structure. In the HNSW graph, each vertex represents a data object, and a directed edge denotes the neighborhood relationship between two objects. The graph is constructed in a bottom-up manner: the base layer 0 contains all data objects, and the subsequent layer $i + 1$ is a sparse subset of the layer i, with the number of vertices decreasing exponentially, thus forming a hierarchical small-world network.

The query procedure consists of two stages. The first stage is a multi-level greedy search that starts from a single entry point in the top layer. It traverses layer by layer, progressively approaching the query vector until an entry point in the base layer is identified. In the second stage, a refined search in the base layer is initiated by a beam search from the entry point, maintaining a candidate set S as a distance-ordered min-heap and a result set R as a max-heap of the K nearest neighbors found so far. Then, it iteratively explores the top point of S until the closest candidate in S is farther away than the farthest neighbor in R, and the top-K objects in R are returned.

IVF [23] is a coarse-grained index structure. During index construction, the dataset O is partitioned into K clusters using the K-means algorithm, and each cluster is represented by its centroid. All data objects are then assigned to the inverted list corresponding to their nearest centroid. For query processing, the query q is first compared against all centroids to identify the $n_{\text{probe}} \geq 1$ closest ones. Subsequently, distance comparisons are performed only on candidates within the nearest n_{probe} inverted lists, in which the K nearest objects are returned.

Both ANNS indexes require calculating the exact distance between each filtered candidate object and the query and comparing these distances. Due to the large number of candidates, repeated distance computations often dominate the overall query overhead. This critical underlying task is widely referred to as *Distance Comparison Operation* (DCO). One solution for this problem is to optimize the index structure and search process to reduce the number of DCOs. For instance, SHG [16] was proposed to optimize HNSW by reducing navigation overhead using compressed vectors in non-base layers and employing piecewise linear models to skip redundant layers during graph traversal. Tribase [50] could reduce DCOs on IVF by precomputing intra-cluster neighbor information to improve the pruning capacity according to the triangle inequality. SuCo [49] proposed to reframe DCOs in ANNS as a more efficient probabilistic subspace collision counting problem. However, these methods still require a large number of DCOs to achieve high recall rates in ANNS. Therefore, to further accelerate ANNS, a more common and practical approach is to delve into fine-grained DCO optimizations.

3 Summary of DCO Optimizations

In this section, we summarize and categorize existing DCO optimization methods. DCO is a fundamental step in ANNS that evaluates whether the distance $\text{dist}(q, o)$ between a query q and a candidate object o is less than or equal to a dynamically updated threshold r. The candidate o is classified as positive if $\text{dist}(q, o) \le r$, and as negative otherwise. Next, we introduce three DCO schemes.

FDScanning: It is a naive DCO implementation that computes the exact distance in all dimensions and compares it against r. Such an exhaustive process results in a time complexity of $O(D)$.

Projection Pruning: Projection maps high-dimensional vectors onto low-dimensional subspaces through linear transformations to alleviate the curse of dimensionality. The projection methods used for ANNS include: *Random Projection* [26], which employs a random orthogonal matrix to achieve homogeneity of variance across dimensions while preserving the distances between objects; and *Principal Component Analysis (PCA)* [27], which maximizes the variance along each projection axis, thereby concentrating the most significant features into a few dimensions.

During the search process, vectors are projected either entirely or partially, and pruning decisions are made based on distances in the projected space. By computing a lower bound or approximation of the distance, an algorithm can safely discard candidates whose distance to the query exceeds the search threshold, thereby avoiding time-consuming full-dimensional computations.

Quantization [17, 20]: It utilizes a finite set of representative vectors, called a codebook, to represent high-dimensional vectors, and the most widely used quantization method is *Product Quantization* (PQ) [23]. PQ divides a D-dimensional vector into M distinct sub-vectors of dimensions D/M. For each subspace, a codebook with k centroids is learned offline by K-means clustering. Each data vector is then quantized by finding the nearest centroid in each subspace codebook and concatenating their respective IDs to form its PQ code. During the search process, an Asymmetric Distance Table (ADT) that contains the distances from the q's sub-vectors to centroids in the corresponding sub-codebook is first computed. The quantized distance to any object o is then estimated by looking up its PQ code: for each subspace, it retrieves the pre-computed distance from the query's sub-vector to the centroid indexed by o's PQ code and sums them up.

According to the DCO scheme used, existing DCO optimization methods are generally categorized into *projection pruning*- and *quantization*-based approaches. Table 1 summarizes the key characteristics of these methods. Next, we discuss the core design ideas of each method.

Table 1: DCO optimization methods evaluated in our paper. In the SIMDized column, we use the symbol "●" to indicate that the original paper has considered and supported SIMD parallel optimization, "◐" to indicate that the original paper does not consider SIMD optimization but the method itself is SIMD parallelizable, and "○" to indicate that the method cannot support SIMD parallelization.

Algorithm	Type	Index	SIMDized
ADSampling [13]	Projection Pruning	HNSW + IVF	◐
Finger [9]	Projection Pruning	HNSW	◐
DADE [51]	Projection Pruning	HNSW + IVF	◐
DDC$_{\text{res}}$ [10]	Projection Pruning	HNSW + IVF	◐
DDC$_{\text{pca}}$ [10]	Projection Pruning	HNSW + IVF	◐
DDC$_{\text{opq}}$ [10]	Quantization	HNSW + IVF	●
PDX-ADS [28]	Projection Pruning	IVF	●
PDX-BOND [28]	Projection Pruning	IVF	●
Flash [47]	Quantization	HNSW	●
RaBitQ [14]	Quantization	HNSW + IVF	●

3.1 Projection Pruning-based Methods

ADSampling [13] is among the earliest algorithms specifically proposed to accelerate DCOs. In the index construction phase, it applies the random projection technique as discussed in §2, which rotates all objects in the dataset isometrically, preserving Euclidean distances while ensuring that the dimensional variance is homogenized for effective sampling.

In the query processing phase, ADSampling employs a strategy that combines incremental dimension sampling with hypothesis testing, as outlined in Algorithm 1. The core idea is to test the hypothesis $H\colon\ \text{dist}(q, o) \le r$. After applying the same transformation to the query q as to the data objects, the test is performed by computing a normalized partial distance dist' from the incrementally sampled dimensions, i.e.,

$$\text{dist}' = \left(\sqrt{\sum_{k=1}^{d}(q_k - o_k)^2}\right) \cdot \frac{\sqrt{D/d}}{1 + \epsilon_0/\sqrt{d}}, \tag{2}$$

where d is the number of dimensions sampled and ϵ_0 is a global parameter that controls the failure rate. Here, dist' provides an approximation of the full-dimensional distance by scaling the partial distance from d sampled dimensions to estimate its expected value in the D-dimensional space, with a constant ϵ_0 to control the estimation error. Specifically, $(1 + \epsilon_0/\sqrt{d})$ makes the distance estimate more conservative, reducing the likelihood of prematurely discarding a positive result. The hypothesis H is rejected with high confidence if dist' exceeds the threshold r, and the candidate is pruned for early termination. Otherwise, the sampling process continues by repeating the test with additional dimensions until the candidate is pruned or its exact distance is decided once all dimensions have been processed.

Beyond this, ADSampling decouples the roles of HNSW's result set R: one subset provides the distance thresholds for DCOs, while the other maintains the search objects. This approach substantially reduces the number of exact distance computations from N_{ef} to k. For IVF, cache optimization is achieved by pre-computing partial distances for the most frequently accessed

Algorithm 1: Hypothesis Testing in ADSampling [13]

```
Input  : Transformed data vector $o'$ and query $q'$, distance threshold $r$, incremental step size $\Delta d$
Output : Test result for if $\mathrm{dist}(o, q) \le r$: 1 for 'yes' and 0 for 'no'
1  Initialize the number of sampled dimensions $d = 0$;
2  while $d < D$ do
3      $d \leftarrow d + \Delta d$;
4      Using the first $d$ dimensions to compute the approximate distance $\mathrm{dist}'$;
5      Conduct a hypothesis testing with the hypothesis $H$ as $\mathrm{dist} \le r$ based on the approximate distance $\mathrm{dist}'$;
6      if $H_0$ is rejected and $d < D$ then
7          return 0;
8      else if $H_0$ is not rejected and $d < D$ then
9          continue;
10     else
11         return 1 if $\mathrm{dist}' \le r$ and 0 otherwise;
```

initial d dimensions before sequential traversal. Notably, all subsequent methods based on ADSampling incorporate these optimized versions of both index structures.

Finger [9] is an optimization method specifically designed for HNSW that accelerates the graph traversal. When evaluating a candidate node o, Finger employs a distance decomposition approach to rapidly estimate the distance between q and the neighbors of o (denoted as p), thereby determining which neighbor to visit next. The distance between q and p is expressed as

$$\mathrm{dist}' = \sqrt{\|q_{\mathrm{proj}} - p_{\mathrm{proj}}\|_2^2 + \|q_{\mathrm{res}}\|_2^2 + \|p_{\mathrm{res}}\|_2^2 - 2q_{\mathrm{res}}^T p_{\mathrm{res}}}. \quad (3)$$

While $\|p_{\mathrm{res}}\|_2^2$ can be fully precomputed during index construction, $\|q_{\mathrm{proj}} - p_{\mathrm{proj}}\|_2^2$ and $\|q_{\mathrm{res}}\|_2^2$ involve low-cost calculations during query processing. To approximate the residual vector dot product $q_{\mathrm{res}}^T p_{\mathrm{res}}$ without direct calculation, it employs a PCA-trained projection to generate LSH signatures for residual vectors, allowing for a fast estimation based on the Hamming distance.

DADE [51] builds upon ADSampling by replacing its random projection with PCA. This can help capture the maximum variance in fewer dimensions and provide a more accurate approximation of the exact distance. DADE's distance estimator is statistically unbiased and given by

$$\mathrm{dist}' = \sqrt{\sum_{k=1}^{d}(q_k - o_k)^2} \cdot \sqrt{\frac{\sum_{k=1}^{D} \lambda_k}{\sum_{k=1}^{d} \lambda_k}} \Big/ (1 + \epsilon_0), \quad (4)$$

where λ_k is the eigenvalue of the k-th principal component, $\sum_{k=1}^{d} \lambda_k$ represents the cumulative energy captured by the first d dimensions, and $\sum_{k=1}^{D} \lambda_k$ is the total energy of the dataset. By prioritizing the most informative dimensions first, DADE can obtain more accurate distance estimates with fewer samples than ADSampling. This leads to earlier and more reliable pruning, minimizing the estimation error in a data-driven manner.

DDC [10], similar to DADE, builds upon ADSampling by replacing random projection with PCA but distinguishes itself by introducing multiple distinct pruning strategies. **DDC$_{\mathbf{res}}$** designs a query-aware distance lower bound estimator based on European distance decomposition. In contrast to DADE's global distance estimator, DDC$_{\mathrm{res}}$ dynamically refines its distance lower bound by incrementally computing the integral component. After processing the first d dimensions, this probabilistic lower bound on the squared distance is given by $\mathrm{dist}' = \|o\|^2 + \|q\|^2 - 2 \cdot \sum_{k=1}^{d} \langle o_k, q_k \rangle - m \cdot \sigma$, where $m \cdot \sigma$ denotes the error bound w.r.t. the remaining dimensions ($d + 1$ to D), σ is the standard deviation of the residual error, and m is a confidence multiplier.

DDC$_{\mathbf{pca}}$ introduces a data-driven approach to distance correction by reformulating the problem as learning a parameterized error boundary. A logistic regression model $L = \mathrm{sign}(m_1 \cdot \mathrm{dist}' + \beta > r)$ is trained using dist' and the threshold r as features. This formulation frames the task as a binary classification problem: Label 0 corresponds to $\mathrm{dist} \le r$ and Label 1 to $\mathrm{dist} > r$. The trained model then determines whether to prune a candidate based on whether this linear combination exceeds a learned offset.

PDX [28] introduces a columnar data layout tightly integrated with the IVF index and the *PDXearch* framework into projection pruning. This layout stores the same dimension from multiple vectors contiguously in memory, a structure that is essential for fully leveraging SIMD instructions to enable dimension-by-dimension SIMD parallelization across multiple vectors and evaluating pruning boundaries for several vectors at once in a dedicated loop, separate from full-distance computation. Furthermore, PDXearch incorporates additional optimizations, such as adaptive step sizing and register optimization, to further improve computational efficiency. Beyond its native exact pruning method for DCO optimization, PDXearch also supports approximate methods, *e.g.,* ADSampling, highlighting its extensibility.

3.2 Quantization-based Methods

DDC$_{\mathbf{opq}}$ [10] demonstrates the versatility of the learned model by applying it to distance approximations derived from PQ. It adapts the model by incorporating PQ-specific features, such as $\mathrm{dist}(q, c')$ and $\mathrm{dist}(o, c')$ along with the threshold r, into the same logistic regression framework as in DDC$_{\mathrm{pca}}$. This provides equally efficient pruning capabilities for a different type of distance approximation.

Flash [47] accelerates HNSW construction through three key innovations: (1) quantized distance computation, (2) reduced random memory access, and (3) optimized SIMD utilization. During indexing, it first applies PCA for dimensionality reduction, followed by PQ for subspace partitioning and codebook generation. For each newly inserted vector o, Flash generates ADT during candidate acquisition (CA) and compresses them using SQ [1] for SIMD register compatibility. Flash employs a novel access-aware memory layout that co-locates neighbor IDs with their corresponding codewords within each subspace, enabling single-instruction loading of complete data batches via SIMD and fully parallelized distance computations. For neighbor selection, Flash uses globally shared symmetric distance tables (SDTs) that are cached for rapid candidate evaluation. During the search stage, it reuses CA-stage optimizations by creating temporary ADTs for the query vector q and implementing SIMD-parallel sub-vector lookups during graph traversal.

RaBitQ [14] is a random quantization method that can control the compression rate and provide theoretical guarantees. RaBitQ normalizes the vectors o and q based on a cluster centroid c as $o_n = \frac{o-c}{\|o-c\|}$ and $q_n = \frac{q-c}{\|q-c\|}$. Therefore, its distance estimation can be decomposed into

$$\begin{aligned}\mathrm{dist}' &= \|o - q\|^2 = \|(o - c) - (q - c)\|^2 \\ &= \|o - c\|^2 + \|q - c\|^2 - 2 \cdot \|o - c\| \cdot \|q - c\| \cdot \langle q_n, o_n \rangle,\end{aligned} \quad (5)$$

where $\|o - c\|$ can be pre-computed and stored during the indexing phase, and $\|q - c\|$ is computed once per query, whose cost is amortized across comparisons with different data vectors. Therefore, the focus should be solely on quantizing unit vectors and estimating their inner products. When constructing a codebook, it generates a codebook by randomly orthogonally rotating a set of uniformly distributed binary vectors. The data vector is taken as the quantization vector closest to the codebook and stored as a binary string of length D. During queries, each dimension of the query vector q is quantized into an integer representation of $\mathrm{B_q}$ bits by performing randomized uniform scalar quantization on the q_n. The distance estimator first uses a 1-bit quantized code to estimate a high-confidence lower bound on the actual distance, then employs a $\mathrm{B_q}$ bit quantized code to estimate a more precise quantized distance, thereby achieving efficient acceleration. Furthermore, the estimator's design is not merely compatible with hardware acceleration but is co-designed for it, achieving high query performance through an optimized implementation on modern SIMD instructions.

3.3 Hardware Acceleration for DCOs

In addition to projection pruning and quantization methods to speed up DCOs at the algorithmic level, hardware acceleration has also been applied to DCOs. These approaches leverage the massive parallelism, custom memory hierarchies, and high-throughput data paths of modern hardware, transforming the inherently sequential or SIMD-based distance calculations into highly concurrent execution flows, thereby significantly boosting ANNS throughput and reducing latency. Examples include [24, 40, 53], which exploit data-level parallelism within a query and task-level parallelism across queries on FPGAs, and [19, 38, 52, 55], which perform index construction and ANNS processing using GPU architectures with fine-grained parallelism. These approaches are not directly comparable to DCO optimizations in our evaluation because they are specific to hardware platforms that differ substantially from modern CPUs.

3.4 Summary

In summary, existing projection pruning-based methods are dedicated to optimizing their hypothesis testing algorithm with the goal of achieving more reliable distance estimation within fewer dimensions to sample. However, most of them are relatively simple to implement, often limited to SIMD instructions for distance calculations, and have not effectively addressed the increased branch prediction errors introduced by pruning decisions. PDX is one of the few exceptions that designs a column storage format and a vector parallelism strategy based on the characteristics of IVF indexing to solve the above problems.

In contrast, existing quantization-based methods have considered optimizations at different levels. At the algorithmic level, there have been ongoing efforts to explore new encoding schemes that balance accuracy and compression ratio. At the implementation level, on the one hand, quantization schemes are carefully designed to align the bit width of SIMD registers, fully exploiting the potential for hardware parallelism; on the other hand, researchers are also committed to designing cache-friendly data layouts to alleviate large-scale random memory access.

4 Experimental Evaluation

4.1 Overview

Evaluation Goals: This paper aims to investigate the practical effectiveness of various distance comparison optimization (DCO) techniques for ANNS in different scenarios, including their potential side effects. Specifically, our evaluation seeks to answer the following research questions:

Q1: What are the efficiency-accuracy trade-offs of various DCO optimizations when applied to IVF and HNSW, and how are they influenced by different factors? (§4.3–§4.5)

Q2: What is the impact of DCO optimizations on query accuracy, and can they maintain high recall rates across different configurations? (§4.6)

Q3: What is the pre-processing overhead incurred by DCO optimizations, and how does this one-time cost weigh against the query performance gains? (§4.7)

Q4: Based on our empirical findings, what design principles can be derived to guide an appropriate selection of DCO optimizations for specific application requirements? (§5)

Experiment Methodology: To answer the aforementioned research questions, we conduct a comprehensive evaluation of mainstream DCO optimizations in ANNS. Our experiments focus on three primary criteria: query efficiency, accuracy, and preprocessing overhead. In addition, we investigate various factors that may affect the effectiveness of DCO optimizations.

To the best of our knowledge, this work is the first to conduct such a fine-grained evaluation. Prior studies primarily emphasize end-to-end performance metrics (*e.g.,* throughput and recall) and lack cross-method, fine-grained comparisons to isolate and analyze the effectiveness of individual DCO techniques. Moreover, most existing evaluations selectively omit critical dimensions, such as preprocessing costs and additional memory overhead, making their results insufficient to guide method selection in different deployment scenarios.

In contrast, we establish a unified benchmarking framework that introduces pruning accuracy as a novel evaluation metric alongside standard QPS-recall trade-offs. We further consider several crucial but previously overlooked factors, including dimensionality, dataset scale, and additional memory overhead introduced by quantization methods. By systematically exploring these factors, we provide a more holistic view of the practical effectiveness of DCO optimizations for ANNS in practice.

4.2 Experiment Setup

Datasets: Our evaluation employs 8 public benchmark datasets, as detailed in Table 2, which span a wide range of sizes, dimensionalities, and modalities. These datasets, comprising both base vectors and queries stored in single-precision format, serve as standard benchmarks for ANNS performance evaluation.

Platform: All experiments were conducted on a server running Ubuntu 20.04.6 LTS, equipped with a 64-core Intel® Xeon® Silver 4314 CPU @ 2.40GHz (x86-64 architecture) and 247 GB of DRAM.

Table 2: Statistics of vector datasets in the experiments.

Dataset	Dimension	#Items	#Queries	Modality
GloVe [41]	25/50/100/200	1,183,514	10,000	Text
DEEP [6]	96	9,990,000	10,000	Image
SIFT/GIST[1]	128/960	1,000,000	10,000	Image
MSong [7]	420	983,185	1,000	Audio
Contriever (CTV) [22]	768	990,000	10,000	Text
InstructorXL (IXL) [44]	768	2,253,000	1,000	Text
OpenAI[2]	1,536	999,000	1,000	Text

[1]http://corpus-texmex.irisa.fr/
[2]https://zilliz.com/ai-models/text-embedding-3-small

The processor supports SSE, AVX2, and AVX-512 SIMD instruction sets. The index construction was parallelized across multiple threads, while query evaluations were executed on a single thread to ensure consistent and reproducible latency measurements.

Evaluation Metrics: We use the following metrics for performance evaluation:

- **Recall@K** evaluates query accuracy, computed as $\frac{|S_{\text{res}} \cap S_{\text{gt}}|}{K}$, where S_{res} denotes the result set returned by the algorithm and S_{gt} represents the ground-truth result.
- **Queries Per Second (QPS)** quantifies the throughput of query processing.
- **False Pruning Rate (FPR@K)** assesses the precision of DCO pruning decisions, calculated as $N_{\text{neg}}/(N_q \times K)$, where N_{neg} denotes the number of ground-truth vectors incorrectly pruned and N_q is the total number of queries.

Algorithms: The DCO optimizations were evaluated with two index structures: HNSW and IVF. Each optimization method was tested on both index types, except those specifically designed for only one type of index. Notably, we included two state-of-the-art index optimization methods, Tribase [50] and SuCo [49], to compare them against DCO optimization methods.

Implementation Details: For HNSW, we set the degree of each node to $M = 16$ and the number of candidates (for construction) to $N_{\text{ef}} = 500$, following common practice [34]. For IVF, we follow the recommendations of the FAISS library [25] to set the number of clusters n_c based on the number of base vectors N. Specifically, we set $n_c = \sqrt{N}$ when $N < 500\,000$, $n_c = 4\sqrt{N}$ when $N > 2\,500\,000$, and $n_c = 2\sqrt{N}$ otherwise. We modify the search ranges of IVF and HNSW by changing the number of probes n_{probe} and the number of candidates (for query processing) n_{ef}.

4.3 Results for DCO Optimizations on HNSW

In this subsection, we evaluate different DCO optimization methods in terms of trade-offs between efficiency and accuracy on HNSW indexes. To ensure a fair comparison, we use QPS at the same recall rate as the main performance metric.

Scalar Search: We first evaluate the performance of each algorithm in a scalar implementation. Figure 3 presents the performance of different DCO optimizations under scalar execution. Flash is excluded as its recall@10 consistently fell below 0.8.

On high-dimensional datasets, *e.g.,* GIST, DADE and DDC$_{\text{res}}$ outperformed ADSampling by 1.47× and 1.62×, respectively. This advantage stems from the fact that vectors projected via PCA have their dimensional components sorted in descending order by variance, enabling more accurate distance estimation with fewer dimensions. The results reveal that at the same recall rates, ADSampling requires approximately 2× more dimensional computations than those of DADE and 4× more than those of DDC$_{\text{res}}$. However, on low-dimensional datasets like DEEP, DDC$_{\text{res}}$ computes fewer dimensions than ADSampling within identical search ranges, yet achieves lower recall. Even when using twice the search range of ADSampling, DDC$_{\text{res}}$ still fails to match ADSampling's recall rate. We suppose that PCA-based pruning methods, while computationally efficient, erroneously exclude actual neighbors near decision boundaries in early phases. The recall degradation necessitates expanded search ranges to compensate for pruning errors.

From an implementation perspective, DADE's distance estimation relies exclusively on a global dimensional distribution array trained during index preprocessing. This approach overlooks potential deviations of individual queries from the global distribution, leading to estimation inaccuracies. For instance, DADE's recall rate decreases significantly when encountering outlier vectors in the GloVe dataset. In contrast, DDC$_{\text{res}}$ incorporates query-specific information into distance estimation. ADSampling demonstrates moderate acceleration across all scenarios, as random projection homogenizes dimensional variance and its proportion-based distance estimation remains highly conservative. Consequently, ADSampling is less prone to erroneous pruning and requires only slight search range expansion.

DDC$_{\text{pca}}$ achieves the best overall performance by requiring computational complexity comparable to DDC$_{\text{res}}$ while leveraging its learned model to identify effective decision boundaries across different sampling dimensions. Notably, unlike the Flash algorithm, which struggles to attain high recall even after reranking, DDC$_{\text{opq}}$ achieves high recall without additional sorting, demonstrating the versatility and practicality of the learned model. Theoretically, the computational overhead of DDC$_{\text{opq}}$ is lower than that of projection pruning, as it only requires accumulating approximate distances between subspaces. However, DDC$_{\text{opq}}$'s memory accesses are nearly 20× those of projection pruning methods, with cache misses reaching up to 39× higher, negating its advantage in faster distance estimation.

Although the original RaBitQ implementation is SIMDized, we implemented and evaluated a scalar version for comparison. RaBitQ introduces an effective integration mechanism for HNSW. It employs 1-bit quantized encoding for preliminary screening at the HNSW entry point and during the search phase. This ensures that only a small number of candidates passing this coarse lower-bound filter proceed to the more expensive bit-distance computation. Compared to hypothesis testing in ADSampling, which requires at least one check per dimension, RaBitQ can make pruning decisions with a single check. Crucially, unlike DDC$_{\text{opq}}$, RaBitQ avoids using lookup tables within HNSW, thereby preventing an increase in random memory access. Consequently, RaBitQ achieves comprehensive performance improvements over all other DCO optimizations under scalar execution.

The findings are summarized as follows. First, approximation inaccuracies inherent in DCO optimization can lead to erroneous results for specific queries. Consequently, DCO optimization methods exhibit lower recall rates than FDScanning under identical search scopes. Second, the acceleration benefits of all DCO optimization methods increase with higher dimensionality. This occurs because higher dimensions enable projection pruning to eliminate more dimensions, whereas quantization reduces the number of floating-point operations. Although DCO optimization methods typically require searching broader ranges to achieve recall rates equal to FDScanning, the total computational cost savings remain substantially greater. Thus, DCO optimization methods consistently outperform FDScanning in efficiency.

SIMDized Search: We then evaluate the performance of different methods under SIMDized execution and examine their interactions with modern CPU features. Figure 4 presents the performance of DCO optimizations when SIMD acceleration is enabled. Surprisingly, the results demonstrate that the baseline HNSW outperforms advanced DCO optimizations. This finding underscores that the superiority of DCO optimizations compared to FDScanning diminishes when confronted with the powerful parallel computing capabilities of modern CPUs.

First, SIMDized HNSW outperforms scalar HNSW by 3.2× on GIST and 2.5× on DEEP on QPS at a recall@10 of 0.95. This improvement stems from the fact that full-dimensional distance computations in HNSW can be significantly accelerated by SIMD

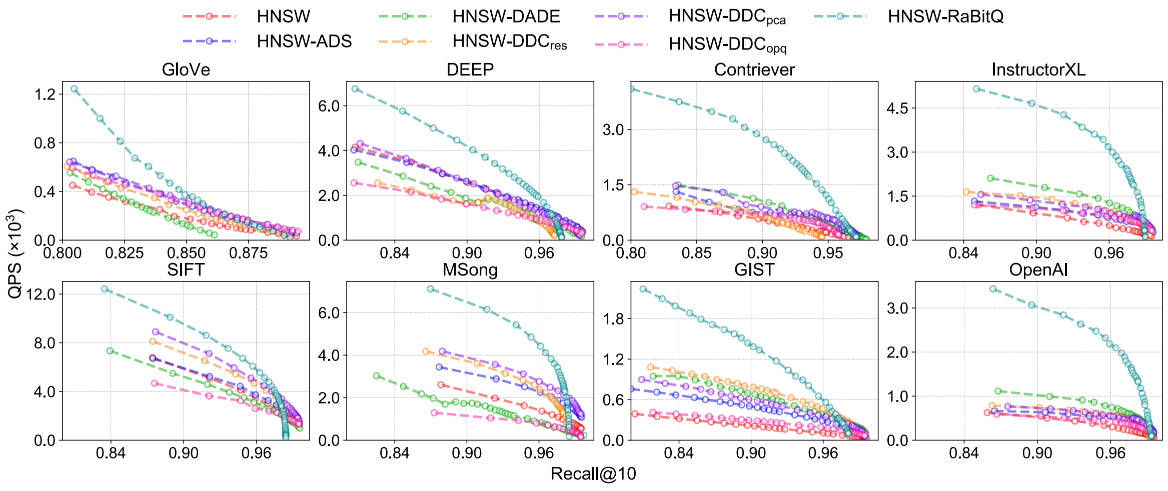


Figure 3: Time-accuracy trade-offs of different DCO optimizations on HNSW with a scalar implementation.

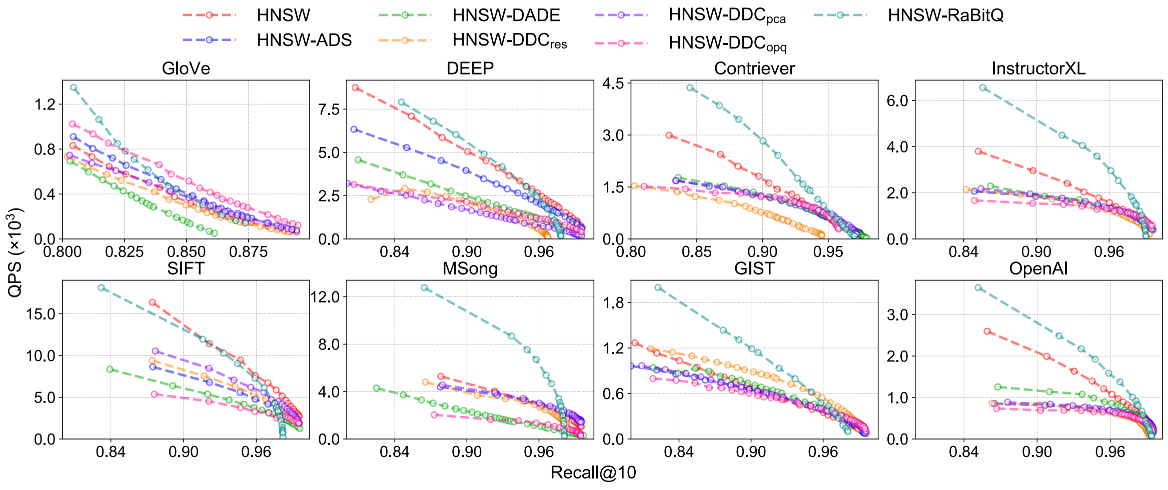


Figure 4: Time-accuracy trade-offs of different DCO optimizations on HNSW with a SIMDized implementation.

instructions, effectively mitigating the primary computational overhead of HNSW. In contrast, projection pruning-based methods benefit less from SIMD acceleration. This is primarily because projection pruning reduces the amount of dimensional computation, leaving limited opportunities for further hardware-level acceleration. Note that the core objective of hardware-level acceleration is to speed up dimensional computations. As a result, at the same recall level, projection pruning-based methods exhibit significantly weaker search performance than the baseline HNSW across almost all datasets (as shown in Figure 4).

Second, projection pruning methods rely on vector projection and threshold comparisons, operations that are not fully amenable to SIMD parallelization. This inherent limitation impedes their parallel efficiency on high-dimensional datasets, thus preventing them from fully capitalizing on SIMD's acceleration capabilities. Moreover, the effectiveness of SIMD instructions depends on a branch-free linear data flow. The hypothesis testing in projection sampling disrupts such a flow, resulting in a significant increase in branch prediction errors. For instance, on the OpenAI dataset, DDC_{res} exhibits a 6.8-fold increase in branch prediction errors compared to the baseline. Consequently, the benefits of projection pruning are often outweighed by the diminished gains of SIMD execution.

Third, although quantization methods generally reduce computation and memory access by mapping 4-byte single-precision floating-point values to more compact data types for each dimension, they fail to effectively accelerate HNSW search. Unlike the integration mechanism adopted in IVF (detailed in §4.4), where quantized distances for an entire cluster can be computed in batches, DDC_{opq} within HNSW must compute distances for each candidate and its neighbors (a small set of size $\leq 2M = 32$) before making traversal decisions. This frequent interruption of parallel computation significantly increases branch prediction errors. Our results indicate that DDC_{opq} incurs nearly 60× more memory accesses and 50× more cache misses than projection pruning, resulting in a greater memory-access overhead than non-SIMDized approaches. Consequently, integrating lookup-table-based methods with HNSW exacerbates the graph index's inherent random memory access problem.

Under SIMD execution, RaBitQ retains its dominance across most datasets. However, the performance advantage of RaBitQ over the baseline is smaller than that of the scalar implementation. This observation underscores a key insight: the substantial advantage of RaBitQ within HNSW stems from the algorithmic design discussed above.

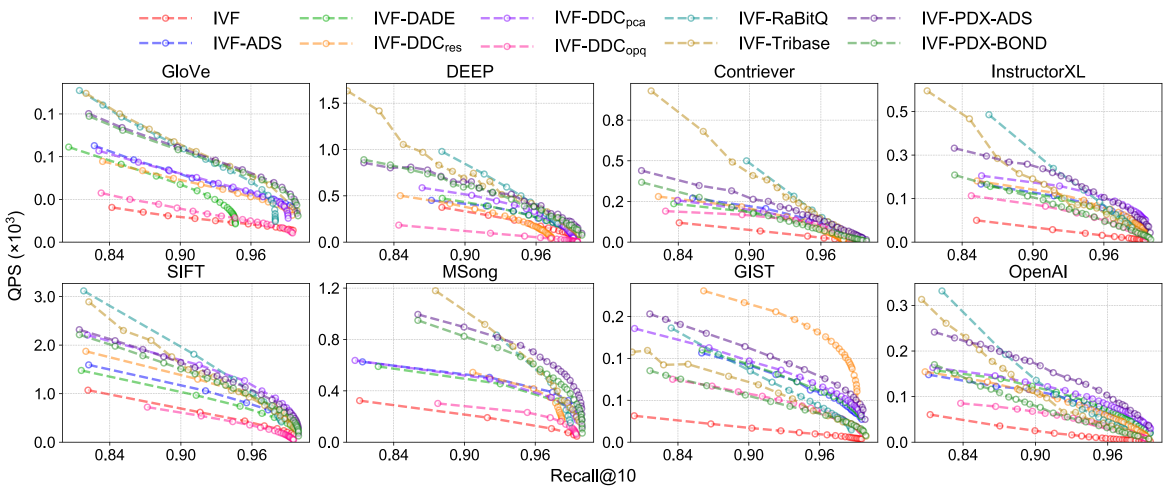


Figure 5: Time-accuracy trade-offs of different DCO optimizations on IVF with a scalar implementation.

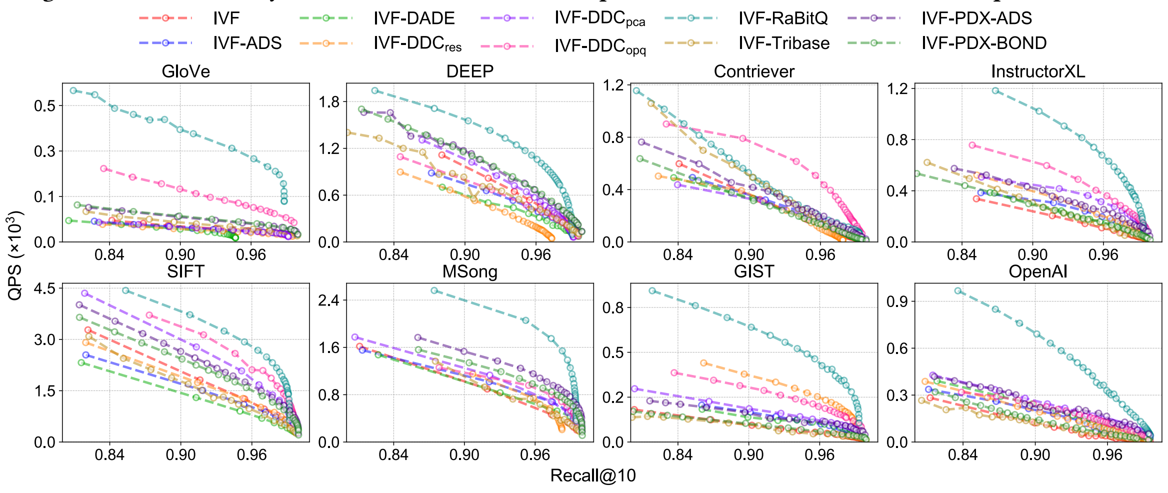


Figure 6: Time-accuracy trade-offs of different DCO optimizations on IVF with a SIMDized implementation.

4.4 Results for DCO Optimizations on IVF

In this subsection, we evaluate the performance of different DCO optimization methods on the IVF index. In contrast to the vector-at-a-time graph traversal paradigm of HNSW, IVF employs coarse-grained data space partitioning that groups vectors into clusters. During query processing, once n_{probe} candidate clusters are selected, all vectors within these clusters are scanned in batches. This exhaustive scanning process requires an IVF-based search to handle a substantially larger number of DCOs compared to HNSW. Consequently, the cost of DCOs becomes the primary performance bottleneck throughout the query process, creating significant opportunities for DCO optimization.

Scalar Search: The performance of different DCO optimizations for IVF under scalar execution is presented in Figure 5. In general, HNSW achieves better search performance than IVF across all datasets. This performance gap reveals that HNSW is more effective at minimizing the candidate set size than IVF's multi-probe approach, as IVF must evaluate a substantially larger number of vectors to achieve the same recall rate. For example, on the GloVe dataset, with a recall rate of 0.85 under scalar search, baseline HNSW outperforms its IVF counterpart by 8.5× in QPS. Our findings, which differ significantly from those observed with HNSW, can be summarized as follows.

First, all DCO optimizations show substantial acceleration across all datasets. Compared with HNSW, almost all evaluated methods significantly outperform the baseline IVF. This provides strong evidence that DCO optimization is more important when DCOs become the bottleneck, and that the additional overhead incurred by optimizations does not offset the acceleration.

Second, a well-designed data layout can significantly improve DCO efficiency. PDX-ADS achieved 1.5× higher performance than ADSampling with horizontal layout at a recall rate of 0.9 on the OpenAI dataset. The primary difference between the two lies in the search framework transition from vector-by-vector to dimension-by-dimension processing. Through the PDXearch framework, PDX-ADS calculates partial dimensions of each vector based on an adaptive step size order under scalar execution and makes pruning decisions in batches. Although it does not utilize automatic vectorization, PDX-ADS significantly enhances the pruning capability of projection pruning methods and reduces hypothesis testing costs. However, PDX-BOND demonstrates excellent pruning capability on low-dimensional datasets but exhibits weaker pruning performance than ADSampling and other methods on high-dimensional datasets such as InstructorXL and GIST. The reason is that PDX-BOND requires substantially higher-dimensional computation than the acceleration achieved through search framework optimization. This validates the effectiveness of ADSampling in dimension reduction via projection and approximate distance estimation based on partial distances.

Unlike the performance improvements observed in HNSW-RaBitQ, a scalar FastScan (a SIMD-based fast implementation for PQ) implementation of IVF-RaBitQ often lags behind many other DCO methods. This performance divergence stems from the fundamental difference in their computational principles. HNSW-RaBitQ relies only on bitwise AND and population count operations. Its SIMD implementation exploits intra-data parallelism, processing 8 data blocks in parallel, which accounts for the moderate performance gap. In contrast, IVF-RaBitQ uses FastScan, a design that is intrinsically bound to SIMD shuffle instructions for register lookup in inter-vector parallelism. SIMD implementations can process 32 vectors in parallel, widening the performance gap relative to scalar implementations. Simulating displacement and masking operations with scalar code requires more instructions and cannot be implemented in parallel. These issues offset the advantages of quantization, leading to a significant performance degradation under scalar execution.

Finally, the co-design of DCO and index optimization schemes can yield greater performance gains. Tribase achieves high QPS in low-recall scenarios, but its performance declines sharply as n_{probe} increases. Although the number of pruned vectors increases exponentially with the doubling of n_{probe}, a substantial number of vectors remain unpruned, which bottlenecks the search performance at high recall. Consequently, DCO optimization is essential, and its outstanding performance underscores its significance as a direction orthogonal to the ANNS index design.

SIMDized Search: Figure 6 shows the performance of IVF with DCO optimizations under SIMDized implementations. Projection pruning-based DCO optimizations generally outperform the baseline IVF. The higher number of DCOs in IVF's search process mitigates the negative impact of branch mispredictions. When amortized across a large number of operations, the performance benefits of projection pruning remain substantial, maintaining a significant advantage over the baseline.

From the perspective of SIMD acceleration, projection pruning methods exhibit limited SIMD acceleration compared to baseline IVF due to dimension reduction computations and the difficult-to-parallelize hypothesis testing loop. In contrast, PDX-ADS consistently outperforms projection pruning methods under SIMDized search when combined with PDXearch's automatic vectorization. On the OpenAI dataset, ADSampling achieves a 1.2× speedup with SIMD, while PDX-ADS achieves a 1.6× speedup, demonstrating the significant acceleration of PDX's automatic vectorization and SIMD instruction set optimization. Similarly, DDC_{opq}'s performance is enhanced by SIMD instructions, performing parallel computation through lookup-based accumulation followed by unified approximate distance estimation and decision-making. However, its performance is limited by random memory access inherent in the lookup method. When switching from SSE to AVX512 instruction sets, the performance gain is negligible.

In contrast, RaBitQ employs a SIMD-based FastScan, achieving the highest performance across nearly all datasets. RaBitQ reshapes batches of 32 vectors into columnar storage format, enabling distance lower-bound estimation while fully leveraging modern CPUs' sequential memory access optimizations. While SIMD instructions are utilized for dimension computations, Tribase's pruning logic remains difficult to parallelize. Specifically, its angle pruning relies on complex floating-point operations (e.g., cosine theorem), which are less amenable to SIMD acceleration.

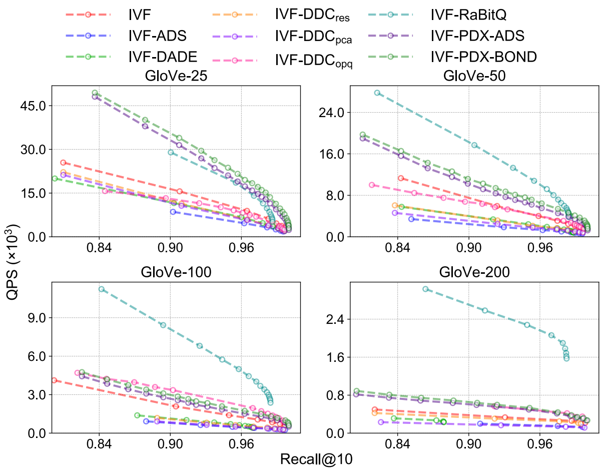


Figure 7: Comparison of different DCO optimizations in the IVF-SIMDized setting with varying dimensions.

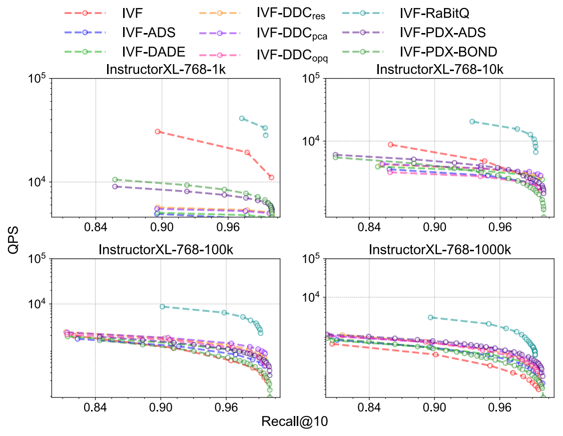


Figure 8: Comparison of different DCO optimizations in the IVF-SIMDized setting with varying data scales.

Furthermore, as the pruning mechanism effectively reduces the number of candidates, the marginal gains from SIMD-accelerated distance calculations are significantly diminished.

4.5 Scalability Test

Effect of Dimensionality: We evaluate the impact of dimensionality on DCO optimization methods on GloVe datasets with 25 to 200 dimensions. The results are shown in Figure 7. We observe that all DCO optimizations gradually enhance their performance advantages as the dimensionality increases, as both projection pruning and quantization can save more computational overhead on higher-dimensional data. We also observe exceptional cases in which certain methods fail to achieve good performance. For example, RaBitQ struggles to converge to high recall rates on GloVe-25 due to insufficient randomness in low-dimensional spaces during high-dimensional orthogonal rotations; DADE experiences mispruning and performance degradation on GloVe-200 for anomalous queries that deviate from global distribution.

Effect of Data Scale: On the InstructorXL dataset, we randomly sample subsets of sizes from 10^3 to 10^6. The results are shown in Figure 8. When the data scale is small, the computational time saved is insufficient to offset the additional overhead resulting

Table 3: Memory usage of $\mathrm{DDC_{opq}}$ and RaBitQ built on the HNSW index. (QV: quantization vectors; QET: quantization error table; LUT: lookup table).

Dataset	HNSW (GB)		QV (MB)		QET (MB)		LUT (KB)	
	$\mathrm{DDC_{opq}}$	RaBitQ	$\mathrm{DDC_{opq}}$	RaBitQ	$\mathrm{DDC_{opq}}$	RaBitQ	$\mathrm{DDC_{opq}}$	RaBitQ
GloVe	1.05	0.43	45.15	220.08	4.52	0	25	1.56
DEEP	4.95	2.61	228.61	990.72	38.11	0	12	0.75
SIFT	0.62	0.26	30.52	125.90	3.81	0	16	1.0
GIST	3.72	0.94	114.44	820.16	3.81	0	120	7.5
MSong	1.67	0.51	98.45	363.24	3.75	0	52.5	3.28
CTV	2.97	0.77	90.63	653.23	3.78	0	96	6.0
IXL	6.76	1.76	206.27	1486.28	8.60	0	96	6.0
OpenAI	5.79	1.41	182.89	1299.27	3.81	0	192	12.0

from fewer DCO executions. As the data scale increases, the number of negative objects increases substantially, making the effect of DCO optimizations more significant. When the size reaches 1 million, all DCO optimization methods outperform the baseline, indicating their applicability to large-scale data.

Memory Overhead Analysis: In addition to time efficiency, we also evaluated the space overhead of DCO optimization methods. For projection-pruning methods, the additional data structures they require are only projection matrices or model parameters, whose sizes are negligible relative to the indexes themselves. For quantization methods, the sizes of quantization vectors and lookup tables are determined by the dimensionality and size of the dataset and can be computed accordingly. Taking the OpenAI dataset ($D = 1536$) as an example, $\mathrm{DDC_{opq}}$ utilizes $M = 192$ subspaces with 8-bit quantization (256 centroids). The additional memory consists of (1) quantization vectors (999000×192 bytes $\approx$ 182.89 MB), (2) quantization error table, which stores per-vector reconstruction distances (999000×4 bytes $\approx$ 3.81 MB), and (3) lookup tables (LUTs) for distance computation ($192 \times 256 \times 4$ bytes $\approx$ 192 KB). To sum up, the space overhead of $\mathrm{DDC_{opq}}$ is small relative to the 5.79 GB memory footprint of the HNSW index.

RaBitQ is built on the FastScan mechanism. In high-precision mode, its LUTs consume up to 12 KB, while its quantization vectors require 1299.27 MB. Notably, RaBitQ eliminates the need for storing quantization error, as it relies on specialized bit-signatures for distance estimation. Compared with $\mathrm{DDC_{opq}}$, the HNSW index of RaBitQ is constructed directly using these quantized vectors instead of the original 32-bit floating-point vectors. This substantially compresses the overall index footprint (*e.g.,* from 5.79 GB to 1.41 GB on the OpenAI dataset). We empirically measured the memory usage of $\mathrm{DDC_{opq}}$ and RaBitQ, and the results align with the above analysis and are presented in Table 3. In general, the additional space overhead of quantization-based DCO optimizations is small relative to index sizes.

4.6 Pruning Accuracy in DCO Optimizations

The accuracy of pruning in DCO optimizations is critical for ANNS performance. Inaccuracy in approximate distance estimations may lead to erroneous pruning decisions. Performance experiments reveal that this issue is particularly pronounced in the HNSW algorithm due to its path-dependent nature, where early errors can lead to suboptimal path selections subsequently and degrade search accuracy. This places high demands on the efficiency and accuracy of the pruning procedure. To provide a more detailed assessment, we report the highest FPR@10 for

Table 4: FPR@10 (%) of different DCO optimizations for HNSW. We exclude PDX-BOND from the comparison because it employs an exact pruning strategy.

Method	CTV	DEEP	GIST	GloVe	IXL	MSong	OpenAI	SIFT
ADSampling	0.21	0.28	0.33	0.14	0.35	0.41	0.41	0.46
DADE	0.29	0.01	0.01	4.66	**0.00**	0.01	0.06	0.02
$\mathrm{DDC_{res}}$	2.51	2.21	0.76	0.37	0.55	1.26	0.63	0.2
$\mathrm{DDC_{pca}}$	0.32	**0.00**	**0.01**	0.33	0.08	**0.11**	0.12	0.15
$\mathrm{DDC_{opq}}$	1.89	0.01	0.09	**0.06**	0.05	0.47	0.28	**0.00**
RaBitQ	**0.17**	0.15	0.77	0.46	0.27	0.57	**0.08**	0.43

each category of DCO optimization schemes on the HNSW algorithm in Table 4. Note that PDX-BOND is excluded from this comparison as it is specifically designed for IVF and performs exact pruning. We classify methods with low recall@10 scores as inaccurate and provide an explanation of their inaccuracy.

Inaccurate Methods: The FPR@10 of Finger decreases with increasing N_{ef}. When integrating the result set decoupling from ADSampling, which reduces the number of full-dimensional computations from N_{ef} to k, into Finger, its recall drops dramatically. These results indicate inaccuracies in Finger's decomposition method for distance estimation and show that its accuracy depends heavily on full-dimensional computations during the initial search stage. Although Flash performs reranking after the HNSW bottom-level search, its recall@10 remains below 0.6.

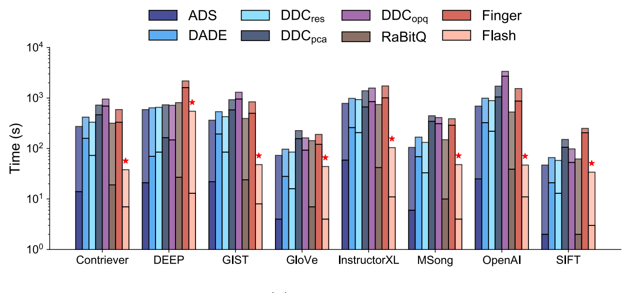
Flash relies solely on the estimated quantization distance for DCO execution, so if the quantization distance exceeds the threshold, it is inserted directly into the maximum heap. Thus, when distance estimation errors are substantial, the search path can deviate, leading to a notable decrease in recall. This confirms the importance of DCO accuracy for HNSW, as discussed earlier.

SuCo's recall@1 exceeds 0.9, but its recall@10 drops to approximately 0.6. This behavior occurs because its subspace collision framework relies on score-based coarse screening, which rapidly filters out non-compact vectors. However, due to the curse of dimensionality, the scores of non-neighbors in high-dimensional spaces may become highly similar to those of actual neighbors. Consequently, the approximate filtering condition loses discriminative power, leading to the omission of numerous neighbors during coarse screening.

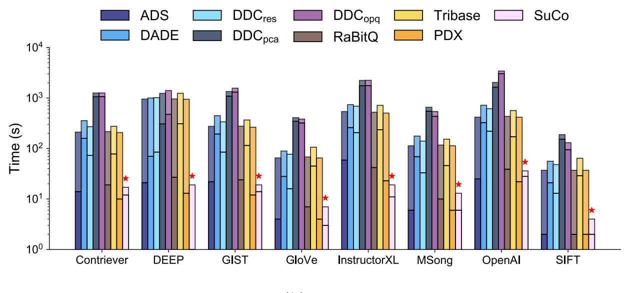
Accurate Methods: We analyze accurate approximate DCO optimizations. We exclude exact pruning methods such as those of Tribase and BOND, which rely on geometric constraints and partial distance bounds rather than approximations. These methods remain preferred choices when recall is the primary concern.

ADSampling consistently maintains a low FPR@10 and a high recall@10, approaching the performance of FDScanning. This performance can be attributed to the combination of incremental dimension sampling and hypothesis testing. However, the method relies heavily on the random projection matrix rather than on the data distribution, which limits its pruning power.

DADE's distance estimator strongly depends on global data statistics λ_k, which may cause incorrect pruning when anomalous queries deviate from these statistics. For instance, on the GloVe dataset, DADE achieves the highest FPR@10 of 4.66%, preventing convergence to high recall rates. In contrast, $\mathrm{DDC_{res}}$ integrates query information into the distance estimation, which theoretically yields more stable false pruning rates. However, it still exhibits considerable erroneous pruning with FPR@10 values exceeding 2% on both the Contriever and DEEP datasets.



(a) HNSW



(b) IVF

Figure 9: Comparison of the HNSW and IVF index construction time for different DCO optimization methods. For each bar, the darker bottom part denotes the pre-processing time, and the lighter top part denotes the indexing time.

Even with the highest-precision (8-bit per dimension) quantization, RaBitQ fails to converge to one recall on certain datasets, *e.g.*, GIST, due to the inherent precision loss of quantization induced by data distributions, which leads to high FPR@10. In contrast, DDC_{pca} and DDC_{opq} successfully achieve high recall through the learned models, demonstrating both the accurate decision limits enabled by learning and the general applicability of this approach across various approximate distance measures.

4.7 Pre-processing Overhead Analysis

Beyond QPS-Recall performance, the additional overhead incurred during index construction and query processing constitutes a critical aspect for a method's practical use. In this subsection, we provide a detailed analysis of these overheads.

Pre-Processing Overhead in Index Construction: The additional costs required for index pre-processing and construction are detailed in Figure 9. All DCO optimizations involve a pre-processing phase to transform and project raw vectors. However, their pre-processing overhead varies significantly.

First, training learned models incurs high overhead. For instance, the pre-processing cost of DDC_{pca} and DDC_{opq} is significantly higher than the index construction itself. However, this upfront investment, which can substantially improve query efficiency and accuracy, is clearly worthwhile. Second, the pre-processing overheads of Finger and Tribase are substantial. Both methods require traversing the index to calculate the second-nearest neighbor for each vector. This introduces a significant time cost comparable to the index construction time.

Other DCO optimizations require only data conversion operations, such as projection calculations or quantization. Compared to index construction, these methods incur negligible overhead while significantly improving query performance. Despite Flash's low query recall rates, its index construction runs one order of magnitude faster than that of other methods. Its compact coding and layout optimization significantly accelerate the massive DCO operations performed during HNSW construction. Furthermore, this layout provides a potential solution for HNSW's random access memory and offers a feasible direction for combining quantization methods with graph-based methods in the future.

Pre-processing Overhead in Query Execution: The cost of pre-processing is a critical factor because it directly impacts runtime search efficiency. As shown in Figure 10, the pre-processing cost of DCO optimization methods with projection operations is high. The projection of the query vector involves matrix-vector multiplication in $O(D \cdot d_p)$ time, where d_p represents the projected dimension. Due to the comparatively smaller number of

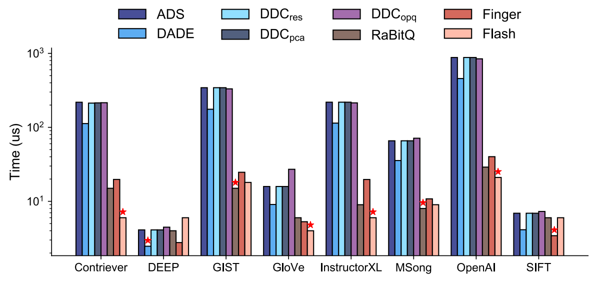


Figure 10: Comparison of the query pre-processing time of each DCO optimization method.

DCOs in HNSW compared to IVF, the projection cost is often too substantial to amortize. For example, on the OpenAI dataset, the projection cost of DDC_{res} in HNSW accounts for 50% of the query time at 90% recall@10, and remains at 14% even at 99% recall@10. In contrast, for DDC_{res} in IVF, it constitutes only 8.9% at 95% recall@10 and drops to 1.7% at 99% recall@10. Thus, to achieve higher efficiency at the expense of lower recall, projection pruning-based DCOs for HNSW might not be good choices.

As shown in Figures 5 and 6, PDX-ADS consistently outperforms PDX-BOND, and the performance gap widens as the dimension increases. This indicates that a one-time pre-processing yields benefits across all DCOs. Moreover, the performance advantage becomes more pronounced with higher recall requirements. Thus, pre-processing represents a trade-off that should be weighed based on user needs and application scenarios.

Compared with projection-based methods, the quantization overhead of RaBitQ and Flash on query vectors is negligible. Tribase and SuCo incur no query-time preprocessing, as all required information is precomputed during index construction. As a result, their query latency is largely insensitive to preprocessing costs, allowing the performance gains from DCO optimizations to be more clearly manifested.

5 Summary of Empirical Findings

In this section, we present the key insights from our empirical evaluation of different DCO optimizations. Based on our findings, we first provide practical guidelines for selecting DCO optimizations under various constraints, as summarized in Figure 11. We then outline promising directions for future research to address the limitations of existing methods.

Although the predominant focus of existing studies in ANNS has been on designing and optimizing new index structures, our

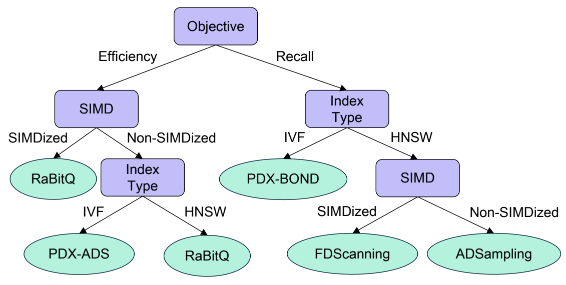


Figure 11: Decision tree for DCO optimization selection.

evaluation demonstrates that optimizing the DCO itself can yield significant performance gains. Furthermore, such micro-level optimizations can be integrated synergistically with macro-level index designs, offering a complementary and highly effective approach to enhancing overall search efficiency.

Guidelines for DCO Optimization Selection: Based on the experimental results, we summarize the following guidelines for the selection of DCO optimizations in different scenarios. First, for scenarios prioritizing high efficiency, we consider two cases regarding whether SIMD is supported. Using SIMDized implementation, RaBitQ is the most suitable choice, which delivers the highest query efficiency on both HNSW and IVF. When SIMD is disabled, the selection becomes index-dependent. For HNSW, RaBitQ is also the most suitable choice; however, for IVF, due to the limited acceleration of FastScan, which is deeply bound to SIMD, PDX-ADS is recommended for its improved pruning efficiency and reduced branch prediction overhead within the PDX architecture. Second, when the main objective is high recall, the selection strategy depends on both the index type and the hardware support. For IVF, PDX-BOND stands out as the optimal method due to its exact pruning mechanism and substantial performance gains. For HNSW, we distinguish between two cases. When SIMD is available, FDScanning achieves superior performance and is directly applicable. When SIMD is unavailable, ADSampling becomes the most suitable option, as it yields a low false pruning rate while greatly improving search efficiency.

Limitation 1: DCO Optimizations Are Decoupled from Vector Index Structures. Currently, DCOs are often optimized in isolation and evaluated under limited conditions. As such, existing DCO optimizations are difficult to implement in real-world systems. This decoupled approach makes it difficult to maximize the performance potential of DCO optimization with respect to specific indices. For instance, IVF's fixed candidate sets stored in contiguous blocks naturally support columnar data layouts and enable batch-level DCO optimizations. This allows the computation of lower bounds for entire groups of vectors, followed by unified pruning decisions, thereby maximizing SIMD parallelism and cache efficiency. In contrast, HNSW's vector-by-vector and path-dependent search requires per-candidate lower-bound estimation with high reliability to guide graph traversal. This vector-by-vector processing pattern fundamentally limits SIMD acceleration to individual candidate calculations and places stricter demands on estimation accuracy to prevent search divergence. Moreover, random memory access in graph indexes is a common problem, and the table-lookup method of quantization can exacerbate its impact. Although Flash offers a promising direction, many issues remain to be addressed.

Limitation 2: Insufficient Cost-Benefit Analysis on Preprocessing Overhead. We note that most of the DCO optimizations are not standalone plugins; they require two layers of pre-processing: (1) an index-specific pre-computation phase to transform data and generate auxiliary structures, and (2) a per-query pre-processing step that itself consumes computational resources. Practitioners must prudently evaluate the overhead: whether the substantial cost of index pre-processing is justifiable for their application, and crucially, whether the latency added by query pre-processing might nullify acceleration benefits, ultimately degrading performance.

Limitation 3: Practicality of DCO Optimizations in Dynamic Scenarios. Existing DCO optimizations are largely designed for static data, lacking the flexibility to handle frequent data insertions or deletions. When data distribution shifts, learned DCO methods are forced into costly global re-processing, such as (1) retraining projection matrices (e.g., PCA or OPQ) to capture the new data distribution, (2) rebuilding auxiliary index structures, and (3) refitting decision models for pruning. These tasks impose prohibitive computational overhead for online processing, severely undermining system availability in dynamic environments. Therefore, there is an urgent need for lightweight adaptive mechanisms that enable DCO to quickly adjust to index updates without repeated pre-processing.

Limitation 4: Design of DCO Optimizations on Specialized Hardware. Although hardware acceleration is not directly comparable to the DCO workloads we measure, its synergy represents a critical future direction. For instance, how can the complex branching logic of pruning algorithms be more efficiently mapped to the configurable logic of FPGAs?

Limitation 5: Generalization to Inner Product and Cosine Distance. Our current evaluation focuses on the squared Euclidean distance, as almost all existing DCO methods are designed and optimized for it. However, we recognize that some modern vector search applications, particularly those driven by LLMs, also adopt other distance measures, *e.g.,* inner product and cosine distance. Expanding DCO optimizations and evaluations to non-Euclidean distances is also a key direction for future work.

6 Conclusion

This paper systematically categorizes and empirically evaluates existing DCO optimization methods for ANNS, demonstrating that DCO optimization is essential for improving the efficiency of ANNS on vector databases and providing scenario-specific decision guidelines for selecting DCO optimization methods. Based on the empirical findings, we note that existing DCO optimizations are limited by their decoupling from index structures and by their costly preprocessing and update overhead. Future work can explore lightweight index update mechanisms to adapt to changes in data distribution and query workloads. Moreover, key challenges remain in leveraging software-hardware co-design for DCOs and proposing DCO optimizations for non-Euclidean distance measures such as inner product and cosine distance.

Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was supported in part by the National Natural Science Foundation of China under Grant No. 92270202 and No. 62472173, National Key Research and Development Program of China under Grant No. 2023YFB4502905, and the Shanghai Sailing Program No. 23YF1410600.

Artifacts

Our code, data, and evaluation results are publicly available at https://github.com/wbh1765091327/DCO_benchmarks.

References

[1] Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore L. Willke. 2023. Similarity search in the blink of an eye with compressed indices. *Proc. VLDB Endow.* 16, 11 (2023), 3433–3446.

[2] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2015. Cache locality is not enough: High-Performance Nearest Neighbor Search with Product Quantization Fast Scan. *Proc. VLDB Endow.* 9, 4 (2015), 288–299.

[3] Akari Asai, Sewon Min, Zexuan Zhong, and Danqi Chen. 2023. Retrieval-based Language Models and Applications. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 6: Tutorial Abstracts)*. Association for Computational Linguistics, 41–46.

[4] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *Proc. ACM Manag. Data* 3, 1, Article 43 (2025), 29 pages.

[5] Artem Babenko and Victor Lempitsky. 2014. The Inverted Multi-Index. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 6 (2014), 1247–1260.

[6] Artem Babenko and Victor S. Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2055–2063.

[7] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*. ISMIR, 591–596.

[8] Jin Chen, Zheng Liu, Xu Huang, Chenwang Wu, Qi Liu, Gangwei Jiang, Yuanhao Pu, Yuxuan Lei, Xiaolong Chen, Xingmei Wang, et al. 2024. When large language models meet personalization: Perspectives of challenges and opportunities. *World Wide Web* 27, 4 (2024), 42.

[9] Patrick Chen, Wei-Cheng Chang, Jyun-Yu Jiang, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. 2023. FINGER: Fast Inference for Graph-based Approximate Nearest Neighbor Search. In *Proceedings of the ACM Web Conference 2023 (WWW)*. Association for Computing Machinery, New York, NY, USA, 3225–3235.

[10] Liren Deng, Peng Chen, Xiaoyun Zeng, Tianyi Wang, Yanjing Zhao, and Kai Zheng. 2024. Efficient Data-Aware Distance Comparison Operations for High-Dimensional Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 18, 3 (2024), 812–821.

[11] Liwei Deng, Hao Sun, Rui Sun, Yan Zhao, and Han Su. 2022. Efficient and Effective Similar Subtrajectory Search: A Spatial-aware Comprehension Approach. *ACM Trans. Intell. Syst. Technol.* 13, 3, Article 35 (2022), 22 pages.

[12] Liwei Deng, Yan Zhao, Jin Chen, Shuncheng Liu, Yuyang Xia, and Kai Zheng. 2024. Learning to Hash for Trajectory Similarity Computation and Search. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4491–4503.

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

[14] 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 (2024), 27 pages.

[15] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 4 (2013), 744–755.

[16] Zengyang Gong, Yuxiang Zeng, and Lei Chen. 2025. Accelerating Approximate Nearest Neighbor Search in Hierarchical Graphs: Efficient Level Navigation with Shortcuts. *Proc. VLDB Endow.* 18, 10 (2025), 3518–3530.

[17] Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE Transactions on Information Theory* 44, 6 (1998), 2325–2383.

[18] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time Personalization using Embeddings for Search Ranking at Airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, New York, NY, USA, 311–320.

[19] Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P. A. Lensch. 2023. GGNN: Graph-Based GPU Nearest Neighbor Search. *IEEE Trans. Big Data* 9, 1 (2023), 267–279.

[20] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. 2000. What is the nearest neighbor in high dimensional spaces?. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*. Morgan Kaufmann, 506–515.

[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*. Association for Computing Machinery, New York, NY, USA, 2553–2561.

[22] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Trans. Mach. Learn. Res.* (2022). https://openreview.net/forum?id=jKN1pXi7b0

[23] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128.

[24] Wenqi Jiang, Hang Hu, Torsten Hoefler, and Gustavo Alonso. 2025. Fast Graph Vector Search via Hardware Acceleration and Delayed-Synchronization Traversal. *Proc. VLDB Endow.* 18, 11 (2025), 3797–3811.

[25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2019), 535–547.

[26] William B. Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. In *Contemporary Mathematics*, Richard Beals, Anatole Beck, Alexandra Bellow, and Arshag Hajian (Eds.). Vol. 26. American Mathematical Society, 189–206.

[27] Ian T Jolliffe and Jorge Cadima. 2016. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374, 2065 (2016), 20150202.

[28] Leonardo Kuffo, Elena Krippner, and Peter Boncz. 2025. PDX: A Data Layout for Vector Similarity Search. *Proc. ACM Manag. Data* 3, 3, Article 196 (2025), 26 pages.

[29] 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 Trans. Knowl. Data Eng.* 32, 8 (2020), 1475–1488.

[30] Defu Lian, Yongji Wu, Yong Ge, Xing Xie, and Enhong Chen. 2020. Geography-aware sequential location recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, New York, NY, USA, 2009–2019.

[31] Yingfan Liu, Hong Cheng, and Jiangtao Cui. 2017. PQBF: I/O-efficient approximate nearest neighbor search by product quantization. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. Association for Computing Machinery, New York, NY, USA, 667–676.

[32] Ying Liu, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. 2007. A survey of content-based image retrieval with high-level semantics. *Pattern Recognit.* 40, 1 (2007), 262–282.

[33] Ce Lyu, Yanhao Wang, Jie Liang, and Minghao Zhao. 2025. LLM-based Dynamic Differential Testing for Database Connectors with Reinforcement Learning-Guided Prompt Selection. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 3814–3818.

[34] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.

[35] Microsoft. 2020. SPTAG: A library for fast approximate nearest neighbor search. https://github.com/microsoft/SPTAG. Accessed: 2025-12-31.

[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. *Proc. ACM Manag. Data* 1, 2, Article 197 (2023), 25 pages.

[37] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, USA, 1933–1942.

[38] Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. 2024. CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4236–4247.

[39] Jingjing Pan, Jianguo Wang, and Guoliang Li. 2024. Vector database management techniques and systems. In *Companion of the 2024 International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 597–604.

[40] Hongwu Peng, Shiyang Chen, Zhepeng Wang, Junhuan Yang, Scott A. Weitze, Tong Geng, Ang Li, Jinbo Bi, Minghu Song, Weiwen Jiang, Hang Liu, and Caiwen Ding. 2021. Optimizing FPGA-based Accelerator Design for Large-Scale Molecular Similarity Search (Special Session Paper). In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.

[41] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1532–1543.

[42] Pinecone Systems, Inc. 2021. Pinecone. https://www.pinecone.io/. Accessed: 2025-12-31.

[43] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative Filtering Recommender Systems. In *The Adaptive Web*, Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl (Eds.). Springer, Berlin Heidelberg, 291–324.

[44] Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2023. One Embedder, Any Task: Instruction-Finetuned Text Embeddings. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics, 1102–1121.

[45] Hao Wang, Tong Xu, Qi Liu, Defu Lian, Enhong Chen, Dongfang Du, Han Wu, and Wen Su. 2019. MCNE: An end-to-end framework for learning multiple conditional network representations of social network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, New York, NY, USA, 1064–1072.

[46] Mengzhao Wang, Boyu Tan, Yunjun Gao, Hai Jin, Yingfeng Zhang, Xiangyu Ke, Xiaoliang Xu, and Yifan Zhu. 2025. Balancing the Blend: An Experimental Analysis of Trade-offs in Hybrid Search. arXiv:2508.01405 [cs.DB]

[47] Mengzhao Wang, Haotian Wu, Xiangyu Ke, Yunjun Gao, Yifan Zhu, and Wenchao Zhou. 2025. Accelerating Graph Indexing for ANNS on Modern CPUs. *Proc. ACM Manag. Data* 3, 3, Article 123 (2025), 29 pages.

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

[49] Jiuqi Wei, Xiaodong Lee, Zhenyu Liao, Themis Palpanas, and Botao Peng. 2025. Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 1, Article 79 (2025), 29 pages.

[50] Qian Xu, Juan Yang, Feng Zhang, Junda Pan, Kang Chen, Youren Shen, Amelie Chi Zhou, and Xiaoyong Du. 2025. Tribase: A Vector Data Query Engine for Reliable and Lossless Pruning Compression using Triangle Inequalities. *Proc. ACM Manag. Data* 3, 1, Article 82 (2025), 28 pages.

[51] Mingyu Yang, Wentao Li, Jiabao Jin, Xiaoyao Zhong, Xiangyu Wang, Zhitao Shen, Wei Jia, and Wei Wang. 2025. Effective and General Distance Computation for Approximate Nearest Neighbor Search. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 1098–1110.

[52] Yuanhang Yu, Dong Wen, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2022. GPU-accelerated Proximity Graph Approximate Nearest Neighbor Search and Construction. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 552–564.

[53] Shulin Zeng, Zhenhua Zhu, Jun Liu, Haoyu Zhang, Guohao Dai, Zixuan Zhou, Shuangchen Li, Xuefei Ning, Yuan Xie, Huazhong Yang, and Yu Wang. 2023. DF-GAS: a Distributed FPGA-as-a-Service Architecture towards Billion-Scale Graph-based Approximate Nearest Neighbor Search. In *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 283–296.

[54] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, et al. 2023. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 377–395.

[55] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate Nearest Neighbor Search on GPU. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1033–1044.

[56] Weichen Zhao, Minghao Zhao, Huiqi Hu, and Weining Qian. 2025. Columnar Formatted Inverted Index for Highly-Paralleled, Vectorized Query Processing. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 1800–1813.