arXiv:2508.17828v1 [cs.DB] 25 Aug 2025

TRIM: Accelerating High-Dimensional Vector Similarity Search with Enhanced Triangle-Inequality-Based Pruning

Yitong Song[1], Pengcheng Zhang[1], Chao Gao[2], Bin Yao[1], Kai Wang[1], Zongyuan Wu[3], Lin Qu[4]
[1] Shanghai Jiao Tong University [2] Zilliz [3] Alibaba Group [4] Taobao
{yitong_song, zhangpc, w.kai}@sjtu.edu.cn, chao.gao@zilliz.com, yaobin@cs.sjtu.edu.cn,
zongyuan.wuzy@alibaba-inc.com, xide.ql@taobao.com

Abstract

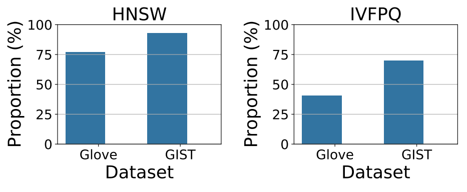
High-dimensional vector similarity search (HVSS) is critical for many data processing and AI applications. However, traditional HVSS methods often require extensive data access for distance calculations, leading to inefficiencies. Triangle-inequality-based lower bound pruning is a widely used technique to reduce the number of data access in low-dimensional spaces but becomes less effective in high-dimensional settings. This is attributed to the "distance concentration" phenomenon, where the lower bounds derived from the triangle inequality become too small to be useful. To address this, we propose `TRIM`, which enhances the effectiveness of traditional **tr**iangle-**i**nequality-based pruning in high-di**m**ensional vector similarity search using two key ways: (1) optimizing landmark vectors used to form the triangles, and (2) relaxing the lower bounds derived from the triangle inequality, with the relaxation degree adjustable according to user's needs. `TRIM` is a versatile operation that can be seamlessly integrated into both memory-based (e.g., HNSW, IVFPQ) and disk-based (e.g., DiskANN) HVSS methods, reducing distance calculations and disk access. Extensive experiments show that `TRIM` enhances memory-based methods, improving graph-based search by up to 90% and quantization-based search by up to 200%, while achieving a pruning ratio of up to 99%. It also reduces I/O costs by up to 58% and improves efficiency by 102% for disk-based methods, while preserving high query accuracy.

Keywords

High-dimensional vector similarity search, Triangle inequality, Approximate k nearest neighbor search, Approximate range search

ACM Reference Format:
Yitong Song[1], Pengcheng Zhang[1], Chao Gao[2], Bin Yao[1], Kai Wang[1], Zongyuan Wu[3], Lin Qu[4]. 2018. TRIM: Accelerating High-Dimensional Vector Similarity Search with Enhanced Triangle-Inequality-Based Pruning. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX



(a) A PG-based method **(b) A PQ-based method**

Figure 1: Proportion of data access and distance calculations

1 Introduction

Recently, high-dimensional vector similarity search (HVSS) plays a crucial role in fields such as information retrieval [47], data mining [48], recommendation systems [59], and AI applications [43]. It aims to find a set of vectors that are similar to a query vector q, which involves two main query types: k nearest neighbor search [19, 41] (kNNS) and range search (RS) [55]. However, due to the "curse of dimensionality" [30], computing exact answers for these queries is cost-prohibitive, which motivates the shift to their approximate versions: approximate k nearest neighbor search (AkNNS) and approximate range search (ARS) [31, 41, 55].

To handle HVSS, proximity graph (PG) [19, 41] and product quantization (PQ) [32] based methods have been developed, exhibiting superior performance and broad application [2, 38, 53, 65]. The PG-based method organizes vector data into a graph structure, connecting each vector to its approximate nearest neighbors, and performs HVSS through greedy traversal on the graph. The PQ-based method encodes vectors into compact PQ codes, allowing rapid distance estimation to the query vector q from these codes. It performs HVSS by identifying numerous vectors with the smallest estimated distances to q as candidate results and refining them according to their exact distances. Despite their differing principles, both suffer from a common performance bottleneck: ***accessing extensive data for distance calculations***. As depicted in Figure 1, on two popular datasets, data access and distance calculations account for over 75% of the query time in a typical PG-based method (i.e., HNSW [41]) due to a large number of vectors traversed on the graph. For IVFPQ [2], a popular PQ-based method, this proportion averages around 50%, primarily due to the refinement phase involving distance calculations for numerous candidates.

Recent studies have focused on reducing the distance calculation cost in HVSS using various distance comparison operations (DCOs) [14, 22, 57]. However, these methods mainly reduce vector dimensionality involved in distance calculations without addressing data access, leading to two key drawbacks: (1) The data access overhead remains significant both when data is stored in memory [1]

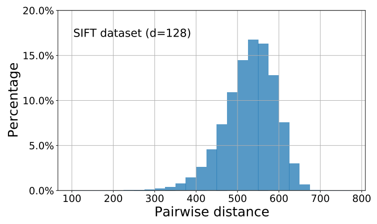


Figure 2: Distance distribution

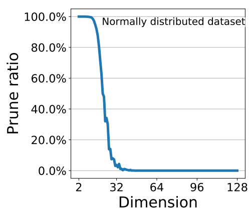


Figure 3: Pruning ratio

and on disk [31, 55]; (2) SIMD can inherently accelerate distance calculations, but these methods progressively test the dimensions used for the computation, which limits SIMD compatibility.

To address both data access and distance calculation overhead, triangle-inequality-based lower bound pruning is widely used in low-dimensional settings, greatly improving the efficiency of various queries by orders of magnitude [16, 40, 45]. In this paper, we explore how to apply such triangle-inequality-based pruning in HVSS. Given a query q and a data vector x, the lower bound lb of the distance $\Gamma(q,x)$ is computed as $lb = |\Gamma(l,q) - \Gamma(l,x)|$, where l is a landmark used to form the triangle $\triangle lqx$. This lower bound can be computed more efficiently than the exact distance, as $\Gamma(l,x)$ is pre-computed and $\Gamma(l,q)$ can be computed at low amortized cost (since it is computed once per query and reused across multiple data). If lb exceeds a certain distance threshold during the HVSS query, x can be pruned without the need for computing $\Gamma(q,x)$, as it is confirmed to be dissimilar to q. Larger lower bounds can enhance the pruning power by excluding more candidates.

However, we observe that directly applying this approach in HVSS is ineffective due to the "*distance concentration*" phenomenon in high-dimensional spaces [17, 46], which causes most vector pair distances to fall within a narrow range, making the lower bounds too small to be useful. As shown in Figure 2, the pairwise distances on a real dataset SIFT with dimension 128 are concentrated between 400 and 700, making the maximum possible lower bound only $700 - 400 = 300$, which approaches the minimum distance in the dataset and thus fails to enable effective pruning. Moreover, our simulations in Figure 3 confirm that the pruning effectiveness of the triangle-inequality-based lower bound declines sharply with dimensionality and becomes negligible beyond 32 dimensions. A common strategy to improve pruning is to use multiple landmarks selected from the dataset to compute a larger bound [15, 25, 45, 62], i.e., $lb = \max_{l\in L} |\Gamma(l,q) - \Gamma(l,x)|$. However, this practice introduces substantial overhead in lower bound calculations, offsetting its benefits of reducing computational overhead in HVSS. Therefore, applying triangle-inequality-based pruning in HVSS faces significant challenges in both obtaining sufficiently tight lower bounds for effective pruning and achieving efficient lower bound calculations.

Motivated by above observations, this paper proposes TRIM, a versatile **tr**iangle-**i**nequality-based operation for high-di**m**ensional similarity searches, which enhances the pruning of traditional triangle inequalities through two main improvements: (1) landmark optimization, and (2) p-relaxed lower bounds, as detailed below.

Landmark optimization. Given the fact that the lower bound derived from a dataset-selected landmark is affected by the distance concentration phenomenon and using multiple landmarks for calculations is inefficient, we propose *generating* (rather than selecting) a *single* (rather than multiple), *optimized* landmark l for each data vector x, aiming to maximizing the lower bound lb of $\Gamma(q,x)$ while minimizing the computational cost of lb. To achieve both, we propose employing the vector represented by x's PQ code as its landmark (rather than itself). For maximizing lb, our strategy places l as close to x as possible (since PQ minimizes the reconstruction error between x and its PQ representation). When pruning a vector x far from q, a landmark l close to x is also likely far from q, thus maximizing $|\Gamma(l,q) - \Gamma(l,x)|$. Ideally, when $\Gamma(l,x)$ is near 0, the lower bound approaches $\Gamma(l,q)$, enabling effective pruning. For minimizing the computational cost of lb, our method allows for rapid calculation of lb by using one landmark per vector and leveraging the efficient distance calculation for $\Gamma(l,q)$ inherent in PQ, along with its SIMD compatibility. This method also offers extra benefits: (1) reduced storage costs for numerous landmarks due to the compact nature of PQ, and (2) seamless integration into HVSS systems, as PQ is widely used for vector compression and indexing.

p-Relaxed lower bounds. To further improve pruning effectiveness in high-dimensional spaces, we introduce *relaxed* lower bounds. We define a function $g(x,q,l)$ as a p-relaxed lower bound function (p-LBF) if it satisfies $P(g(x,q,l) \le \Gamma(q,x)^2) = p$, meaning it has a confidence level of $p (0 \le p \le 1)$ to be below $\Gamma(q,x)^2$. Compared to the strict one, p-LBF provides larger lower bounds to improve pruning, though it also decreases query accuracy proportionally to p. We propose a simple yet effective p-LBF: $g(x,q,l) = (\Gamma(l,q) - \Gamma(l,x))^2 + 2\gamma\Gamma(l,x)\Gamma(l,q)$, enhancing pruning by adding a positive term $2\gamma\Gamma(l,x)\Gamma(l,q)$ to triangle inequality's strict lower bound. The factor γ adjusts the degree of relaxation, with larger γ yielding lower p. γ can either be manually tuned or estimated using our proposed method. Our method establishes a correlation function between γ and p by combining theoretical analysis with empirical fitting, enabling γ to be decided for any given p. Surprisingly, even with $p = 1$, we can still yield a large γ value, greatly enhancing pruning with almost no loss of accuracy. Allowing for a slight reduction in accuracy can further boost pruning.

Given the simplicity and ease of integration of TRIM, we then explore its potential as a basic operation to accelerate popular HVSS solutions, including both memory-based PG and PQ series methods and disk-based methods. We thoroughly investigate how TRIM is integrated into each category of methods to enhance query efficiency, offering a comprehensive demonstration of its effectiveness via theoretical analysis and experimental validation. Notably, compared to other DCOs, TRIM excels in its *simplicity*, *versatility* (applicable to both memory- and disk-based AkNN and ARS queries), *SIMD compatibility*, *effectiveness* and *efficiency*. Our key contributions are summarized as follows.

- **Identifying challenges of applying the triangle inequality to HVSS.** We explore the application of triangle-inequality-based pruning to HVSS, and point out the inherent challenges both theoretically and experimentally (Section 2).
- **A versatile operation with improved pruning effect.** We propose TRIM to enhance the triangle-inequality-based pruning in HVSS through two key improvements: landmark optimization and p-relaxed lower bounds, offering a user-friendly way to tune the relaxation degree (Section 3).

- **Enhancing memory-based HVSS methods.** We integrate TRIM into PG- and PQ-based methods for both AkNNS and ARS queries. Experiments show up to 90% improvement in PG-based methods with a pruning ratio of 99%, and up to 200% enhancement in PQ-based methods (Sections 4 and 6).
- **Enhancing disk-based HVSS methods.** We further integrate TRIM into disk-based HVSS methods and present a new method called tDiskANN. By optimizing the data layout and query algorithm, tDiskANN achieves a 58% reduction in I/O cost and improves efficiency by 102% (Sections 5 and 6).

2 Preliminaries

2.1 Two Types of Queries in HVSS

k **Nearest Neighbor Search (kNNS)** is a fundamental query type for high-dimensional vectors. Given a finite set of vectors $\mathcal{D}$ in a d-dimensional Euclidean space E^d, a query vector $q \in E^d$, and an integer k, the goal is to identify k vectors in $\mathcal{D}$ most similar to q. Formally, kNNS returns a subset $\mathcal{R}_{knn} \subseteq \mathcal{D}$ such that $|\mathcal{R}_{knn}| = k$, and $\forall_{v \in \mathcal{R}_{knn}} \forall_{o \in \mathcal{D} \setminus \mathcal{R}_{knn}} \Gamma(v, q) \leq \Gamma(o, q)$, where $\Gamma(.,.)$ represents the Euclidean distance function.

Due to the inherent difficulty of exact kNNS in high-dimensional spaces [19, 41], recent research [19, 22, 31, 39, 41, 55] has focused on the approximate kNNS problem, known as AkNNS. By relaxing the requirement for exact results, these studies have significantly improved query efficiency. The accuracy of AkNNS is typically measured by the query recall, defined as

$$Recall@k = \frac{|\mathcal{R}_{knn} \cap \mathcal{R}'_{knn}|}{k},$$

where $\mathcal{R}_{knn}$ and $\mathcal{R}'_{knn}$ represent the exact and approximate result sets, respectively.

Range Search (RS) is another basic query for high-dimensional vectors, aiming to find all vectors in $\mathcal{D}$ that are within a given distance *radius* from a query vector q. Mathematically, RS returns a subset $\mathcal{R}_{range} \subseteq \mathcal{D}$ such that $\forall v \in \mathcal{R}_{range}, \Gamma(v, q) \leq radius$.

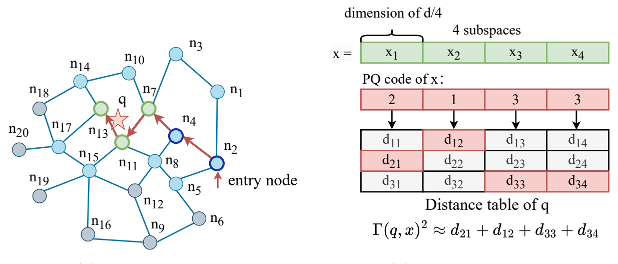
To improve efficiency, approximate range search (ARS) has been a focus of research [55]. The accuracy of ARS is measured using average precision ($AP@e\%$), defined as

$$AP@e\% = \frac{|\mathcal{R}'_{range}|}{|\mathcal{R}_{range}|},$$

where $\mathcal{R}_{range}$ and $\mathcal{R}'_{range}$ represent the exact and approximate result sets, respectively, and $e\%$ indicates the percentage of vectors that fall within the query range.

Note that although this paper focuses on Euclidean distance, our method naturally extends to other metrics, such as inner product (IP), cosine similarity, and angular distance, via data preprocessing (e.g., vector normalization and augmentation). Such transformations are widely adopted in prior work, including RaBitQ [20, 23] and FAISS [2], to convert between distance metrics. However, our method cannot be applicable to non-metric distances that violate the triangle inequality (e.g., asymmetric metrics).

PG- and PQ-based methods [36, 38] are widely employed for AkNNS and ARS queries. In the next section, we review these methods and analyze their performance bottlenecks.



(a) The PG method **(b) The PQ method**

Figure 4: PG- and PQ-based methods

2.2 Review of PG- and PQ-based Methods

PG-based methods, such as HNSW [41] and NSG [19], employ a graph structure to organize and retrieve data vectors. As illustrated in Figure 4a, in this structure, each node represents a vector, connected to its approximate nearest neighbors in different directions. We call a node v a neighbor of a node p if it is connected by p.

During the AkNNS query process, a dynamic candidate queue $\mathcal{S}$ of size ef ($ef \geq k$) is maintained to track ef nodes currently closest to the query q. The parameter ef balances query efficiency and accuracy. A best-first search then iteratively starts from an entry node. At each iteration, it accesses currently located node's neighbors and computes their distances to q. $\mathcal{S}$ is then updated to reflect ef closest nodes to q and the currently located node is marked as visited. In the next iteration, the nearest unvisited node in $\mathcal{S}$ is selected as current node, and its neighbors are examined in a similar manner. The process continues until all nodes in $\mathcal{S}$ have been visited, at which point the first k nodes in $\mathcal{S}$ are returned. For the ARS query, the search proceeds similarly, but an unbounded result queue R is kept to store nodes with distances to q less than *radius*. The search continues until no nodes with distances less than *radius* are found, at which point the nodes in R are returned.

Note that different PG structures may vary in their query algorithms, such as the differences in the number of queues for searching and termination conditions [19, 41], but their core principle remains the same: traversing the graph to identify nodes closest to q until no closer nodes are found.

PQ-based methods segment d-dimensional data into m subspaces, clustering each into C groups. The vector is encoded into an m-dimensional code, where each dimension—a positive integer between 1 and C—reflects the group to which the sub-vector belongs in the subspace. As shown in Figure 4b, a d-dimensional vector x is mapped to a 4-dimensional code, with a "2" in the first dimension indicating that x_1 belongs to the second group in the first subspace.

During query execution, a distance table is first computed, capturing the squared distances from q to each cluster centroid. Specifically, q is divided into m subspaces, and the squared distances are calculated between each subspace component q_i and the corresponding centroids. The squared distance between each data vector x and q is then estimated by summing the distances from q to the centroids x belongs to in each subspace. As illustrated in Figure 4b, $\Gamma(q, x)^2$ is estimated by summing d_{21}, d_{12}, d_{33}, and d_{34}, where d_{ji} represents the squared distance between q_i and the centroid c_j assigned to x_i. This approach enables efficient calculations from q to all data vectors, requiring only $Cm\, d/m$-dimensional distance calculations and mn table lookups, where n is the number of data vectors. For AkNNS queries, the vectors with the k smallest estimated distances are identified and returned. For ARS queries, vectors are returned if their estimated distances fall within *radius*.

However, as noted in [55, 66], compressing vectors into PQ codes for calculations introduces severe errors, which greatly impair query accuracy. To mitigate the issue, a refinement phase is introduced to identify k' vectors with the smallest estimated distances as candidates, where k' is set much larger than the expected result count. The exact distances for these vectors are then computed to accurately determine which satisfy the query condition.

Performance bottlenecks of PG- and PQ-based methods arise from accessing extensive data for distance calculations, much of which does not contribute to the search process or query results [10, 22]. For PG-based methods, each iteration requires visiting all neighboring nodes for distance calculations, even though some are not included in the candidate queue $\mathcal{S}$. For example, in Figure 4a, an AkNNS query with $k = 3$ traverses all neighbor nodes (colored blue) of the nodes on the traversal path, each of which undergoes the distance calculation. However, certain "negative" neighbors, such as n_3, are far from q and will not be reflected by $\mathcal{S}$, making these calculations redundant. In PQ-based methods, achieving high query accuracy often requires a large value of k', leading to extensive distance calculations during the refinement phase. However, if certain vectors are clearly far from q, these calculations can be avoided. Moreover, determining an appropriate value for k' is challenging, particularly for ARS queries, as estimating the number of vectors within a *radius* from q is non-trivial, potentially leading to unnecessary data access and distance calculations.

2.3 Analysis of Applying Triangle Inequality

To reduce unnecessary data access and distance calculations, triangle-inequality-based lower bound pruning is commonly used in low-dimensional settings [15, 16, 35, 40, 45], which can be naturally extended to HVSS. Specifically, for a query vector q, a data vector x, and a landmark l, the lower bound lb of $\Gamma(q, x)$ is computed from the triangle $\triangle lqx$ as $lb = |\Gamma(l, q) - \Gamma(l, x)|$. Here, $\Gamma(l, x)$ can be precomputed, and $\Gamma(l, q)$ is computed once per query and reused across multiple data vectors, resulting in a low amortized cost. If lb fails to meet the inclusion criteria for the candidate or result queues, x can be pruned without further distance calculations.

However, we find that these lower bounds often become too small to be effective due to the "*distance concentration*" phenomenon in high-dimensional spaces [17, 46], where distances between data points tend to be similar as dimensionality increases. As a result, for a landmark l following the same distribution of q and x, the distances $\Gamma(l, q)$ and $\Gamma(l, x)$ are nearly equal, causing lb to approach 0 and rendering pruning ineffective, as proven in Theorem 1.

Theorem 1. *Given a query vector q, a data vector x, and a landmark l, all in E^d, $|\Gamma(l, q) - \Gamma(l, x)|$ tends to 0 as $d \to \infty$.*

Proof. Assume that q, x, and l are random vectors in $\mathbb{E}^d$, with components q_i, x_i, and l_i being independent and identically distributed (i.i.d.) random variables with finite variance σ^2. The squared distance is defined as $\Gamma(l, q)^2 = \sum_{i=1}^{d}(l_i - q_i)^2$. Since $l_i - q_i$ are i.i.d. with mean 0 and variance $2\sigma^2$, we have

$$\mathbb{E}[\Gamma(l, q)^2] = 2d\sigma^2, \quad \mathrm{Var}[\Gamma(l, q)^2] = 8d\sigma^4.$$

By Chebyshev's inequality, for any $\varepsilon > 0$,

$$\Pr\left[\left|\Gamma(l, q)^2 - 2d\sigma^2\right| \geq \varepsilon d\right] \leq \frac{8\sigma^4}{\varepsilon^2 d}.$$

Thus, $\Gamma(l, q)^2$ concentrates around $2d\sigma^2$ as $d \to \infty$.

Using the delta method or smoothness of the square root function, this implies that $\Gamma(l, q) = \sqrt{\Gamma(l, q)^2}$ concentrates around $\sqrt{2d}\sigma$ as $d \to \infty$, and similarly for $\Gamma(l, x)$. Therefore, the difference $|\Gamma(l, q) - \Gamma(l, x)|$ tends to zero in probability as $d \to \infty$. □

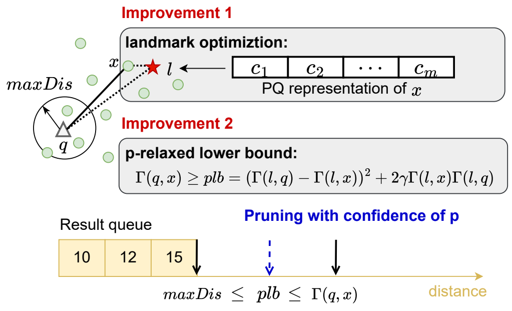


Figure 5: TRIM overview

Our simulations, shown in Figure 3, also confirm that the pruning effectiveness of triangle-inequality-based lower bounds diminishes significantly as dimensionality increases, becoming negligible beyond 32 dimensions. Since HVSS typically handles vectors with dimensions exceeding 96 [31, 41, 55], straightforward application of triangle-inequality-based pruning proves to be ineffective.

To improve pruning effectiveness, current methods typically select a batch of landmarks, compute lower bounds for a data point using each of these landmarks, and retain the largest one for pruning [15, 25, 45, 62]. These landmarks are either selected randomly from the dataset or chosen using heuristic algorithms to maximize their differences [15, 25, 62]. However, when applied to HVSS, these methods face challenges in both *efficiency* and *effectiveness*. First, the computational overhead of calculating lower bounds scales with the number of landmarks, leading to inefficiency. Second, the largest lower bound often remains too loose to be effective, as these landmarks are selected from the dataset and still influenced by the distance concentration phenomenon. To address these limitations, we introduce `TRIM`, a versatile **t**riangle-**i**nequality-based operation for high-dimensional **s**imilarity **s**earches in the next section.

3 The `TRIM` Operation

Figure 5 presents an overview of `TRIM`, which enhances traditional triangle-inequality-based pruning through two key improvements: landmark optimization (Section 3.1) and p-relaxed lower bounds (Section 3.2). `TRIM` uses the PQ representation of each data vector as its unique landmark for the lower bound calculation and relaxes the strict lower bound from the triangle inequality to enable more aggressive pruning with a controllable confidence level p. We then first introduce these two improvements, followed by a detailed description of `TRIM`'s overall process in Section 3.3.

3.1 Landmark Optimization

Given that computing lower bounds using multiple landmarks reduces efficiency, and landmarks selected from the dataset lead to ineffective pruning, we propose *generating* a *single optimized* landmark for each data vector x, aiming to maximize its lower bound distance $|\Gamma(l,q)$ -$\Gamma(l,x)|$. Given that $\Gamma(l,q)$ must be computed on the fly, which makes the computation of $\Gamma(l,q)$ a challenge when assigning a landmark for each data vector, our goal also involves minimizing the computational cost of $\Gamma(l,q)$.

To achieve both, we propose using the PQ method [32] to generate landmarks, where the vector represented by x's PQ code serves as its landmark (rather than itself). For example, as shown in Figure 4b, the landmark for x is generated by combining four $d/4$-dimensional sub-vectors, each corresponding to its cluster centroid in the respective subspace. This approach offers advantages in both minimizing the lower bound computational cost and maximizing lower bounds, as detailed below.

Benefits for lower bound computation. Using PQ representations as landmarks enables efficient computation of $\Gamma(l,q)^2$ by looking up the distance table T, which is generated on the fly at a low cost of $O(Cd)$. Specifically, $\Gamma(l,q)^2$ is exactly computed as $d_{c_1 1}+d_{c_2 2}+\cdots+d_{c_m m}$, where c_i is the i-th value of the PQ code. With n' vectors to be processed during querying, the total time complexity of computing $\Gamma(l,q)$ for n' landmarks is only $O(Cd+n'm)$. This can also be accelerated using SIMD, further enhancing efficiency.

Benefits for maximizing lower bounds. We first formalize the optimal landmark generation problem in Problem 1, and then show that our method offers an effective solution for this problem.

Problem 1. ***Optimal Landmark Generation**: Given a data vector $x \in \mathcal{D}$ and a query set Q, generate a landmark l for x that minimizes the total Mean Squared Error (MSE) between the actual distances from x to queries in Q and the lower bounds derived from l:*

$$MSE(x) = E\left[\sum_{q\in Q}\left(|\Gamma(l,q)-\Gamma(l,x)|-\Gamma(q,x)\right)^2\right] \tag{1}$$

Since the locations of query vectors are often unpredictable, we approach this problem heuristically: If a query vector q can be located anywhere, the landmark l should be positioned to either maximize or minimize $\Gamma(l,x)$, so that the difference $|\Gamma(l,q)-\Gamma(l,x)|$ can be maximized and closest to $\Gamma(q,x)$. Given that the goal of HVSS pruning is to eliminate vectors far from q, it is more effective to position l such that $\Gamma(l,x)$ is minimized. This is because when x is far from q, a landmark l close to x is likely far from q, thereby increasing the difference between $\Gamma(l,q)$ and $\Gamma(l,x)$. Conversely, maximizing $\Gamma(l,x)$ often provides no significant advantage for pruning, as a distant landmark is likely also far from most query vectors. Given this insight, we reduce Problem 1 to Problem 2.

Problem 2. ***Query-Agnostic Landmark Generation**: Given a vector data $x \in \mathcal{D}$, the objective is to find a landmark l that minimizes the MSE between the vector data and its landmark:*

$$MSE(x) = E[\Gamma(l,x)^2] \tag{2}$$

We present Deduction 1, which shows that the optimization task in Problem 2 aligns with the PQ optimization task, making our method provide an effective solution for enlarging lower bounds.

Deduction 1. *The task in Problem 2 aligns with that of PQ.*

Proof. PQ maps a data vector x to a code $pq(x)$, with its quality measured by $MSE(x) = E\left[\Gamma(x,pq(x))^2\right]$. The smaller the $MSE(x)$, the better the quantizer, and PQ is designed to minimize $MSE(x)$ as much as possible. By treating the vector represented by $pq(x)$ as the landmark l, Problem 2 becomes equivalent to the PQ's task. □

An alternative solution to Problem 2 is clustering the dataset and using the closest cluster centroid to x as its landmark. However, this may degrade landmark quality for vectors distant from the centroids. Increasing the number of clusters W helps but adds significant pre-calculation overhead. In contrast, our method provides high-quality landmarks without extra pre-calculation costs. By dividing the data space into m subspaces and clustering each with C centroids, PQ generates C^m potential landmarks, much larger than W (when $C = W$), increasing the likelihood of a close match between data vectors and landmarks. The pre-calculation cost for our method is $O(mCn\frac{d}{m}t)$ when using k-means [33] clustering with t iterations, which is comparable to $O(Wndt)$ for the clustering method.

Additional benefits. Our approach offers further advantages: (1) Significant storage savings, as landmarks are stored as compact PQ codes. (2) Seamless integration into HVSS systems, as PQ is a widely-implemented method for indexing and compressing vectors.

In addition to above analysis, we conduct experiments, detailed in Section 6.5, to show that our strategy significantly outperforms other widely-used methods for generating landmarks.

3.2 p-Relaxed Lower Bounds

Another improvement is to *relax* the lower bound derived from the triangle inequality to provide a larger lower bound and enhance pruning. First, we define the strict lower bound function as follows.

Definition 1. ***Strict Lower Bound Function (LBF)**. Given three vectors x, q, and l, $f(x,q,l)$ is a strict lower bound function if $f(x,q,l) \leq \Gamma(q,x)^2$ always holds.*

According to the triangle inequality, we obtain that $f(x,q,l) = (\Gamma(l,q)-\Gamma(l,x))^2$ is an LBF. Next, we introduce the concept of p-relaxed lower bound function.

Definition 2. ***p-Relaxed Lower Bound Function (p-LBF)**. Given three vectors x, q, and l, $g(x,q,l)$ is a p-relaxed lower bound function if it satisfies $P(g(x,q,l) \leq \Gamma(q,x)^2) = p$.*

p-LBF assigns a confidence level p ($0 \leq p \leq 1$) to the lower bound, meaning it is expected to be less than $\Gamma(q,x)^2$ with confidence level p. A p-LBF provides a larger lower bound than an LBF, improving the pruning effect, though it reduces the query accuracy proportionally to p. We introduce a simple yet effective p-LBF, as shown in Equation 3. This p-LBF enlarges the lower bound by adding a positive term $2\gamma\Gamma(l,q)\Gamma(l,x)$ to the LBF derived from the triangle inequality, where $\gamma \geq 0$ is a tunable parameter.

$$g(x,q,l) = (\Gamma(l,q)-\Gamma(l,x))^2 + 2\gamma\Gamma(l,q)\Gamma(l,x) \tag{3}$$

This p-LBF is proposed inspired by the cosine theorem, i.e.,

$$\Gamma(q,x)^2 = \Gamma(l,q)^2 + \Gamma(l,x)^2 - 2\Gamma(l,q)\Gamma(l,x)\cos\theta, \tag{4}$$

where θ is the angle between the edges lq and lx. Rearranging the cosine theorem yields:

$$\Gamma(q,x)^2 = (\Gamma(l,q) - \Gamma(l,x))^2 + 2\Gamma(l,q)\Gamma(l,x)(1-\cos\theta),$$

which forms the prototype of our p-LBF. The closer γ is to $1-\cos\theta$, the closer the obtained lower bound is to the true squared distance $\Gamma(q,x)^2$. If the probability of $\gamma \leq 1-\cos\theta$ equals p, i.e., $P(\gamma \leq 1-\cos\theta) = p$, we achieve a confidence level p such that $g(x,q,l) \leq \Gamma(q,x)^2$, as formalized in Lemma 1. A higher p leads to lower γ, yielding a looser bound that improves accuracy but reduces pruning and query efficiency.

Lemma 1. *The confidence level p of the p-LBF in Equation 3 is governed by γ, expressed as $p = P(\gamma \leq 1-\cos\theta)$.*

Next, we propose the way for setting γ based on a user-defined p, though for simplicity, γ can also be manually adjusted.

Determining γ for a given p. According to Lemma 1, the confidence level p corresponds to the probability $P(\gamma \leq 1-\cos\theta)$. We formalize the task of finding the cumulative distribution function (CDF) of $1-\cos\theta$ in Problem 3.

Problem 3. ***CDF of*** $\mathbf{1-\cos\theta}$*: Let $X = (X_1,\ldots,X_d) \in \mathcal{D}$ be a fixed d-dimensional vector, $L = (L_1,\ldots,L_d)$ be its associated landmark, and $Q = (Q_1,\ldots,Q_d)$ be a query vector. Determine the CDF of $1-Z$, where $Z = \cos(X-L, Q-L)$.*

To illustrate the challenge of characterizing the distribution of $1-Z$, we begin by assuming that each component of Q is independently drawn from the standard normal distribution, though our approach can be used in general cases. We first analyze the distribution of Z^2, and then transform it to find the distribution of $1-Z$. The following theorem offers a key step in deducing the distribution of Z^2.

Theorem 2. *Given two vectors X' and L', if $\|L'\| = \|L\|$ and $\cos^2(X-L,L) = \cos^2(X'-L',L')$, then the distributions of $\cos(X-L,Q-L)$ and $\cos(X'-L',Q-L')$ are identical.*

Proof. Given $\|L'\| = \|L\|$ and $\cos^2(X-L,L) = \cos^2(X'-L',L')$, there is a rotation matrix R such that $R(L) = L'$ and $R(X-L) = X'-L'$, preserving the angles and magnitudes. Since the distribution of Q is rotation-invariant, the distributions of $R(Q)$ and Q are identical. Therefore, $\cos(X'-L',Q-L') = \cos(R(X-L),R(Q-L))$ has the same distribution as $\cos(X-L,Q-L)$. □

Let $h_1 = \frac{(X-L)L}{\|X-L\|}$ and $h_2 = \sqrt{\|L\|^2 - h_1^2}$. The expression of Z^2 is given by the following theorem:

Theorem 3. *The expression for Z^2 is given by $Z^2 = \frac{A}{A+B+C}$, where $A = (Q_1+h_1)^2$, $B = (Q_2-h_2)^2$, and $C = \sum_{i=3}^{d} Q_i^2$.*

Proof. (Sketch.) Let $X' = (0,h_2,0,\ldots,0)$ and $L' = (-h_1,h_2,0,\ldots,0)$. By Theorem 2, the distribution of $\cos(X-L,Q-L)$ is identical to that of $\cos(X'-L',Q-L')$. Thus, to find the distribution of Z^2, we analyze the distribution of $\cos^2(X'-L',Q-L')$:

$$\cos^2(X'-L',Q-L') = \frac{(Q_1+h_1)^2}{(Q_1+h_1)^2 + (Q_2-h_2)^2 + \sum_{i=3}^{d} Q_i^2}. \quad (5)$$

With $A = (Q_1+h_1)^2$, $B = (Q_2-h_2)^2$, and $C = \sum_{i=3}^{d} Q_i^2$, we express Z^2 as $\frac{A}{A+B+C}$. □

Theorem 3 shows that Z^2 is the ratio of three independent components with different distributions. Since h_1 and h_2 are often nonzero, such ratios generally do not have simple closed-form distributions, which makes the exact derivation of Z^2 (and Z itself) challenging [49, 52]. To address this, we propose two empirical fitting methods: one for queries following a standard normal distribution (e.g., the `NYTimes` dataset), and another for queries without a significant distribution pattern (e.g., the `GloVe` dataset). The first strategy suits scenarios where data is standardized, as in many machine learning tasks. For example, latent variables in Variational Autoencoders (VAEs) are often modeled as standard normal, and synthetic datasets used for benchmarking typically follow this distribution as well. The second strategy targets real-world embeddings (e.g., images, texts, or user behavior), which often exhibit skewness, multimodality, or heavy tails, requiring more flexible fitting.

For queries without a clear distribution pattern, we directly sample a representative subset of data vectors and queries, compute the values of $1-Z$, and fit the CDF. However, this approach is computationally expensive, as it requires sampling many d-dimensional vectors and performing extensive distance calculations.

For queries following a standard normal distribution, we leverage the fact that $A \sim \chi_1(h_1^2)$, $B \sim \chi_1(h_2^2)$ (non-central chi-squared distributions), and $C \sim \chi_{d-3}$ (a chi-squared distribution with $d-3$ degrees of freedom). This allows us to reduce computational costs through the following steps:

- Generate random variants following the distributions $\chi_1(h_1^2)$, $\chi_1(h_2^2)$, and χ_{d-3}, respectively.
- Compute the values of Z^2 using Equation 5 for all variants, and then fit the CDF $F_{Z^2}(z)$ of Z^2.
- Obtain the CDF $F_{1-Z}(y)$ of $1-Z$ using Theorem 4.

Theorem 4. *$F_{1-Z}(y)$ can be expressed in terms of $F_{Z^2}(z)$ as:*

$$F_{1-Z}(y) = \begin{cases} \frac{1}{2} - \frac{1}{2}F_{Z^2}((1-y)^2), & 0 \leq y \leq 1 \\ \frac{1}{2} + \frac{1}{2}F_{Z^2}((1-y)^2), & 1 < y \leq 2. \end{cases}$$

Proof. The CDF of $1-Z$ is given by $F_{1-Z}(y) = P(1-Z \leq y) = P(Z \geq 1-y)$. For $0 \leq y \leq 1$, this is equivalent to $\frac{1}{2}P(Z \leq -(1-y)) + \frac{1}{2}P(Z \geq 1-y)$ since $Z = \cos\theta$, which translates to $\frac{1}{2}P(Z^2 \geq (1-y)^2)$. Using $F_{Z^2}(z)$, we have $P(Z^2 \geq (1-y)^2) = 1 - F_{Z^2}((1-y)^2)$, and thus $F_{1-Z}(y) = \frac{1}{2} - \frac{1}{2}F_{Z^2}((1-y)^2)$. For $1 < y \leq 2$, the relation similarly gives $F_{1-Z}(y) = \frac{1}{2} + \frac{1}{2}F_{Z^2}((1-y)^2)$. □

Instead of directly fitting $F_{1-Z}(y)$, we first fit $F_{Z^2}(z)$ and transform it. This approach significantly reduces computational costs, as it only requires sampling three one-dimensional distributions, applying Equation 5, and performing a simple transformation.

Figure 6 shows an example of a resulting CDF computed for a data vector x on the `NYTimes` dataset. Given a parameter p, the corresponding γ can be directly obtained from the CDF. Notably, even with $p = 1$, a large γ (e.g., 0.7) can still be achieved, enabling a much tighter lower bound with negligible loss in query accuracy. In practice, setting $p = 1$ by default avoids aggressive pruning and removes the need for manual tuning.

It is also important to clarify that we generally set a global γ value for the entire dataset, rather than a local one for each data vector, to reduce pre-calculated and storage overhead. This is achieved by

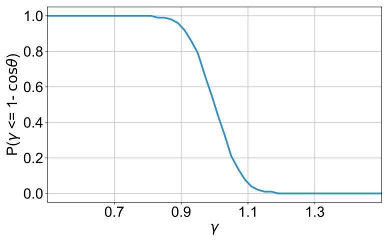


Figure 6: The CDF for a given vector x and its landmark

calculating CDFs for a representative subset of data vectors and retaining the lowest γ value for a given p.

3.3 Processes of TRIM

Building on these two improvements, we propose `TRIM`, with its preprocessing and query processing phases detailed below.

The preprocessing phase involves two steps: (1) For each data vector x, we generate its landmark l using the PQ method, storing both its PQ code and distance $\Gamma(l, x)$ for future use. We also save the cluster centroids produced in the PQ process for computing the distance table. (2) For a representative subset of data vectors, we compute and store their CDFs to determine a global γ for a given p on the fly. If p remains fixed across queries, γ can be pre-computed and stored, eliminating the need to store CDFs.

The query processing involves three operations: In the initial step, (1) construct a distance table T by calculating the squared distances from q to the centroids in each subspace. T is used to compute $\Gamma(l, q)$, by summing the squared distances from q to the centroids corresponding to l in each subspace; (2) based on a given p, compute the lowest γ value (if not pre-computed) using the stored CDFs to establish the p-LBF for the search. (3) In the iteration step, as shown in Figure 5, when reaching a vector x, compute a p-relaxed lower bound plb for pruning, where $\Gamma(l, q)$ is retrieved from T, and $\Gamma(l, x)$ is directly read from the stored data. If plb exceeds the maximum distance limit, x is pruned—with confidence level p—without computing the exact distance.

`TRIM` can be easily integrated into popular memory-based PG and PQ series methods, as well as disk-based methods. Compared to existing PQ-integrated methods (e.g., HNSW+PQ, IVFPQ, and PQ+Vamana), `TRIM` differs in three aspects: (1) it uses a carefully designed p-LBF instead of PQ distances for estimation, (2) it prunes using lower bounds rather than estimated distances, and (3) it adopts optimized query algorithms and data layout (see Sections 4 and 5). The integration of `TRIM` is detailed in the following sections.

4 TRIM for Memory-Based Methods

This section details how to integrate `TRIM` into memory-based HVSS methods, categorized into PG-based and PQ-based methods.

4.1 Integration for PG-based Methods

Recall that PG-based methods utilize a graph structure to perform queries via a best-first search. Below, we describe how to integrate `TRIM` into AkNNS and ARS queries, respectively.

AkNNS queries. Algorithm 1 outlines the query process using `TRIM`. The algorithm follows other Distance Comparison Operations (DCOs) [22, 57], utilizing three queues for querying: search

```
Algorithm 1: AkNNS_For_PG ($e, q, k, ef, p$)
  Input: entry node $e$, query vector $q$, parameter $k$, candidate queue
         size $ef$, and confidence level $p$
  Output: $k$ nearest nodes of $q$
1 Initialize unbounded search queue $\mathcal{S} = \{(plb_e, e)\}$, candidate
    queue $C = \{(\Gamma(q,e), e)\}$ (size $ef$), result queue
    $\mathcal{R} = \{(\Gamma(q,e), e)\}$ (size $k$), and visited set $\mathcal{V} = \{e\}$;
2 $maxDis \leftarrow$ the max. distance between $q$ and the nodes in $\mathcal{R}$;
3 $maxCanDis \leftarrow$ the max. distance between $q$ and the nodes in $C$;
4 Calculate the distance table $T$ and $\gamma$ (if not pre-computed);
5 while $\mathcal{S} \neq \emptyset$ do
6     $(plb_x, x) \leftarrow$ pop the node nearest to $q$ in $\mathcal{S}$ and its lower
       bound;
7     if $plb_x > maxCanDis$ and $|C| = ef$ then
8         break;
9     for each unvisited neighbor $v$ of $x$ do
10        $\mathcal{V}$.add($v$);
11        $plb_v \leftarrow$ compute the relaxed lower bound of $\Gamma(q,v)$;
12        if $|C| < ef$ or $plb_v < maxDis$ then
13            $\mathcal{S}$.add($plb_v, v$);
14            $C$.add($\Gamma(q,v), v$); Resize $C$ and update $maxCanDis$;
15            $\mathcal{R}$.add($\Gamma(q,v), v$); Resize $\mathcal{R}$ and update $maxDis$;
16        else
17            if $plb_v < maxCanDis$ then
18                $\mathcal{S}$.add($plb_v$,$v$);
19                $C$.add($plb_v$,$v$); Resize $C$ and update $maxCanDis$;
20 return $\mathcal{R}$;
```

queue $\mathcal{S}$, candidate queue C, and result queue $\mathcal{R}$, sorted by lower bounds, hybrid distances (lower bounds and exact distances), and exact distances, respectively. During initialization (Lines 1–4), these queues and the visited set $\mathcal{V}$ are initialized with an entry node, while $maxDis$ and $maxCanDis$ track the maximum distances from q to nodes in $\mathcal{R}$ and C. The distance table T and parameter γ are also computed (if not pre-computed) for lower bound calculations. During iteration (Lines 5–19), for each unvisited neighbor v of the node x with the smallest lower bound in $\mathcal{S}$, its relaxed lower bound plb_v for $\Gamma(q, v)$ is computed. If $|C| < ef$ or $plb_v < maxDis$, the distance $\Gamma(q, v)$ is computed, and v is added to $\mathcal{S}$, C, and $\mathcal{R}$. Otherwise, the exact distance is skipped, and if $plb_v < maxCanDis$, v and its lower bound are added to $\mathcal{S}$ and C for searching. The algorithm terminates when C contains ef nodes, and the lower bound plb_x of the current node x exceeds $maxCanDis$ (Lines 7–8).

ARS queries. The ARS query algorithm is similar to the AkNNS algorithm, with a key difference that $\mathcal{R}$ is unbounded. When evaluating a neighboring node v, Line 12 is modified: if $|C| < ef$ or plb_v is not larger than the threshold $radius$, the exact distance is computed. Node v is added to $\mathcal{R}$ only if its exact distance is within the threshold; otherwise, no exact distance calculation is performed. The termination conditions are consistent with those of AkNNS.

Time complexity and performance analysis. Let n' be the number of nodes accessed by the original algorithm, and let a be the pruning ratio of `TRIM`. The original algorithm has a complexity of $O(n'd)$. With `TRIM`, the complexity becomes $O(Cd+mn'+(1-a)n'd)$, where $O(Cd + mn')$ accounts for the pruning cost (i.e., computing the distance table and lower bounds), and $(1-a)n'd$ represents the remaining distance calculations. The cost of computing γ is omitted, as it can be precomputed or efficiently evaluated.

As noted in [57], the performance of DCOs depends primarily on pruning cost (i.e., distance estimation overhead) and pruning ratio. The PQ-based method has been shown to offer the lowest pruning costs. However, its direct pruning ratio without applying the triangle inequality is very low ($\leq$ 50%). TRIM bridges this gap, achieving a pruning cost comparable to PQ while significantly improving the pruning ratio to 99% (see Section 6.2). More importantly, with SIMD optimization enabled, many DCOs fail to outperform the original algorithm due to their poor SIMD compatibility and the reduced cost of exact distance calculations under SIMD. Conversely, TRIM is SIMD-friendly and achieves an ultra-high pruning ratio, providing a significant performance advantage even when distance calculations become cheaper.

Space complexity. Integrating TRIM into PG-based methods introduces additional storage, consisting of four components: (1) distances of n vectors to their landmarks, (2) landmark identifiers (i.e., PQ codes) for n vectors, (3) centroids in each subspace (requiring Cm space), and (4) CDFs or a γ value (constant space f). The total additional space complexity is $O(n + mn + Cm + f)$. Compared to the PG index, which typically requires over 16 connections per vector [41], these additional storage costs are generally modest. TRIM only adds a float value and a bit-stored PQ code (8 bits for $C = 256$) per vector, along with small, amortized constant overheads.

4.2 Integration for PQ-based Methods

Recall that PQ-based methods encode all data vectors into PQ codes, use the distance table T to estimate distances to q (termed PQ distances), and identify k' ($k' \geq k$) vectors with the smallest PQ distances as candidates. In AkNN queries, the top-k vectors with the smallest exact distances to q are returned, while ARS queries return vectors whose exact distances fall within *radius*.

When integrating TRIM, the p-LBF is used to estimate distances and provide lower bounds for pruning. Specifically, traditional PQ-based methods approximate the data vector x using its PQ-code-represented landmark l, i.e., $\Gamma(q,x)^2 \approx \Gamma(q,l)^2$. In contrast, TRIM estimates the distance as:

$$\Gamma(q,x)^2 \approx \Gamma(l,q)^2 + \Gamma(l,x)^2 - 2\Gamma(l,q)\Gamma(l,x)(1-\gamma).$$

In AkNNS queries, a result set $\mathcal{R}$ of size k is maintained, populated with the first retrieved vectors. The maximum exact distance in $\mathcal{R}$ is tracked by *maxDis*. For each subsequent vector, if its estimated distance (i.e., lower bound) is less than *maxDis*, its exact distance is computed and added to $\mathcal{R}$; otherwise, it is pruned without distance calculation. In ARS queries, the process is similar, except that $\mathcal{R}$ is unbounded, and the exact distance is computed only when the lower bound is less than the threshold *radius*.

Our method provides several advantages. First, it dynamically determines the number of candidate results during the query process, eliminating the need to pre-define k'. This is especially beneficial for ARS queries, where determining the number of results in advance is challenging. Second, it removes the need for an explicit refinement step. By providing efficient pruning, our method significantly decreases the number of data vectors that need to be accessed for distance calculations.

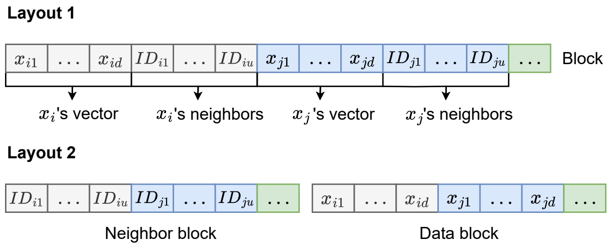

Layout 1
x_{i1} ... x_{id} ID_{i1} ... ID_{iu} x_{j1} ... x_{jd} ID_{j1} ... ID_{ju} ...
Block
x_i's vector
x_i's neighbors
x_j's vector
x_j's neighbors
Layout 2
ID_{i1} ... ID_{iu} ID_{j1} ... ID_{ju} ...
x_{i1} ... x_{id} x_{j1} ... x_{jd} ...
Neighbor block
Data block

Figure 7: Data layouts in disk

5 TRIM for Disk-Based Methods

Current disk-based methods like DiskANN [31] and Starling [55] combine PQ- and PG-based methods for HVSS. They store PQ codes in memory, while data vectors and the PG index reside on disk. As shown in Layout 1 of Figure 7, each disk block stores data vectors and their neighbor IDs (positioned adjacent to the data vector). Starling further saves block reads by placing neighboring nodes (e.g., x_i and x_j) in the same block. During AkNNS queries, a fixed-size search queue $\mathcal{S}$ (sorted by PQ distances) is used for navigation and a result queue $\mathcal{R}$ (sorted by exact distances) is kept. The algorithm starts at an entry node, marks it as visited, and accesses the block containing the node's data vector and neighbor IDs. It computes the node's exact distance to update $\mathcal{R}$ and calculates PQ distances for a subset of neighbors in the block to update $\mathcal{S}$. The search iterates by selecting the nearest unvisited node in $\mathcal{S}$ until all nodes are visited. For ARS queries, multiple rounds of the AkNNS-like process are run to progressively explore nodes within *radius*.

Limitations of existing methods. **First**, the tightly-coupled storage of data vector and neighbor IDs requires them to be loaded together, resulting in unnecessary I/O costs. For example, nodes far from the query q typically do not contribute to updating $\mathcal{R}$ and are used only for navigation, making accessing their data vectors wasteful. **Second**, Starling's layout loses effectiveness for the data dimension $d > 1000$, where typical 4KB blocks can store only one vector and its neighbors, negating the advantage of co-locating neighboring nodes. Even with larger page sizes (8KB or 16KB), each block can store only a few vectors, limiting the effectiveness of neighbor co-location. **Third**, ARS queries suffer from inefficiency due to multiple rounds of exploration for the number of results.

Improvements using TRIM and other optimizations. To address these limitations, we propose tDiskANN, which integrates TRIM into existing methods with enhanced data layout. Specifically, we first decouple the storage of neighbor IDs and data vectors, as illustrated in Layout 2 of Figure 7. In this layout, neighbor IDs and vectors are stored in separate blocks—neighbor blocks and data blocks—while still co-locating neighboring nodes within their respective blocks. This design provides two key benefits: (1) it allows access to only the neighbor IDs, avoiding unnecessary data loading, and (2) it keeps the benefit of co-locating neighboring nodes in the neighbor blocks, even when $d > 1000$, as the neighbor count per node is typically below 40 [41]. Under this data layout, TRIM is then employed to identify nodes whose data blocks do not need to be read. By reducing data access at the algorithmic level, our method improves efficiency largely independent of page size.

Algorithm 2: AkNNS_Of_tDiskANN (e, q, k, ef, p)

```
   Input: entry node $e$, query vector $q$, parameter $k$, search queue size
          $ef$, confidence level $p$
   Output: $k$ nearest nodes of $q$
1  Initialize search queue $\mathcal{S} = \{(pqdis_e, e)\}$ (size $ef$), result queue
   $\mathcal{R} = \emptyset$ (size $k$), and visited set $\mathcal{V} = \{e\}$ ;
2  $maxDis$ ← the max. distance between $q$ and the nodes in $\mathcal{R}$;
3  Calculate the distance table $T$ and $\gamma$ (if not pre-computed);
4  while $\mathcal{S} \neq \emptyset$ do
5      $(pqdis_x, x)$ ← pop the node nearest to $q$ in $\mathcal{S}$;
6      if $x$'s neighbor IDs is not in the LRU cache then
7          access and cache the block $B$ containing $x$'s neighbors;
8      else
9          access $x$'s neighbor IDs from the cache;
10     for each unvisited neighbor $v$ of $x$ do
11         $\mathcal{V}$.add($v$);
12         $\mathcal{S}$.add($pqdis_v$, $v$); Resize $\mathcal{S}$;
13     $plb_x$ ← compute $x$'s relaxed lower bound based on $pqdis_x$;
14     if $|\mathcal{R}| = k$ and $maxDis < plb_x$ then
15         continue;
16     access the data block $B'$ containing $x$ if not already read;
17     for each node $b$ in $B'$ do
18         if $|\mathcal{R}| < k$ or $\Gamma(q, b) < maxDis$ then
19             $\mathcal{R}$.add($\Gamma(q, b)$, $b$);
20             Resize $R$ and update $maxDis$;
21 return $\mathcal{R}$;
```

The overall AkNNS algorithm is described in Algorithm 2. In each iteration, we access the nearest unvisited node x in $\mathcal{S}$, retrieve its neighbor IDs from the neighbor block or cache (Lines 5-9), and compute the PQ distances of its neighbors to update $\mathcal{S}$ (Lines 10-12). Note that the cache we use stores only neighbor IDs, which differs from DiskANN, where several neighbor IDs and data vectors are pre-fetched [31]. Next, TRIM determines whether to read the data vector of x (Lines 13-15). If $plb_x > maxDis$ and $|\mathcal{R}| = k$, the data block is pruned without disk access. Otherwise, the data block is accessed, and the exact distances of vectors in the block are computed to update $\mathcal{R}$ (Lines 16-20). In the ARS algorithm, the result queue $\mathcal{R}$ is unbounded and the data block for a node x is accessed only if its lower bound is within *radius*. Our ARS algorithm can dynamically adjust the number of candidate results in a single round, eliminating the need for multiple rounds of exploration.

6 Experiments

6.1 Experimental Setup

Datasets. As summarized in Table 1, we use widely adopted benchmark datasets for evaluation [22, 31, 41, 54], covering a diverse range of data sizes, dimensions, and sources. Different datasets are selected for memory-based and disk-based experiments. For memory-based evaluations, the GloVe, NYTimes, Tiny5M, and GIST datasets are used for most evaluations, while the SIFT10M dataset is used to assess scalability. Among these, NYTimes contains Gaussian-distributed vectors, whereas the others lack clear distribution patterns. GloVe and NYTimes are originally measured using angular distance; following the preprocessing in Section 2.1, we convert them to Euclidean space for evaluation. For disk-based experiments, we follow Starling [55] by evaluating query performance within a fixed-sized data segment, adjusting the number of vectors accordingly for the Cohere and OpenAI datasets. We also include the 100M-scale SIFT dataset to further assess the scalability.

Table 1: Statistics of datasets

Storage Type	Dataset	Dimension	#Vectors	#Queries	Source	Data Size
Memory	GloVe	100	1,183,514	10,000	Texts	0.4 GB
	SIFT10M	128	10,000,000	10,000	Images	1.2 GB
	NYTimes	256	290,000	10,000	Texts	0.3 GB
	Tiny5M	384	5,000,000	1,000	Images	7.2 GB
	GIST	960	1,000,000	1,000	Images	3.6 GB
Disk	SIFT100M	128	100,000,000	10,000	Images	12.0 GB
	Cohere	768	2,000,000	1,000	Texts	6.0 GB (A data segment)
	OpenAI	1536	1,000,000	1,000	Texts	

Compared methods. In our in-memory experiments, we integrate TRIM into two widely used HVSS algorithms: HNSW [41] (PG-based) and IVFPQ [2] (PQ-based), resulting in two variants: tHNSW and tIVFPQ. tHNSW is compared with HNSW and its DCO variants, namely HNSW_ADS, HNSW_PCA, HNSW_OPQ, and HNSW_RaBitQ. The DCOs is selected following the benchmark [57]: ADS (i.e., ADSampling [22]) and PCA [57] are top-performing transformation-based techniques that reduce computation via dynamic dimensionality reduction, while OPQ [24] and RaBitQ [20, 23] represent state-of-the-art distance estimation methods for pruning. We choose HNSW_OPQ over HNSW_PQ for its theoretical advantages and superior performance in our preliminary experiments. tIVFPQ is compared with IVFPQ and a clustering-based method, Tribase. To further accelerate our method, we integrate FastScan [3, 5], resulting in tIVFPQfs, which is compared against other FastScan-based baselines: IVFPQfs and IVFRaBitQfs. For disk-based experiments, we compare tDiskANN with two state-of-the-arts: DiskANN [31] and Starling [55].

Metrics. We evaluate HVSS query performance in terms of both efficiency and accuracy. For efficiency, we report queries per second (QPS) in in-memory experiments, and both mean I/Os and QPS in disk-based settings [31, 55]. For accuracy, we follow prior work [55], using average recall (Recall@k) for AkNNS queries and average precision (AP@e%) for ARS queries, as defined in Section 2.1. To assess the effectiveness of DCOs, we measure the pruning ratio, as well as the number of estimated and exact distance calculations (EDC and DC, respectively). ADS and PCA compute pruning ratios at the dimension level, whereas TRIM, Tribase, OPQ, and RaBitQ operate at the vector level. Additionally, we assess the tightness of TRIM 's lower bound by analyzing the ratio and difference between the bound and the actual distance, as detailed in Section 6.5.

Parameter setting. Unless otherwise stated, we adopt the following default parameters. For HNSW and tHNSW, we set $M = 16$ and $efConstruction = 500$, following [41]. For the TRIM component, we set $p = 1$ by default (with γ auto-derived), and configure PQ parameters for landmark generation based on [2] and empirical results: $C = 256$, and $m = d/8$ for GIST, $d/4$ for other datasets (see Figure 17). For IVFPQ and tIVFPQ, we use $C' = 4096$ clusters and $C = 256$ subspace clusters, following [2, 23, 57]. The number of subspaces m is chosen based on Figure 17 and standard PQ configurations [2].

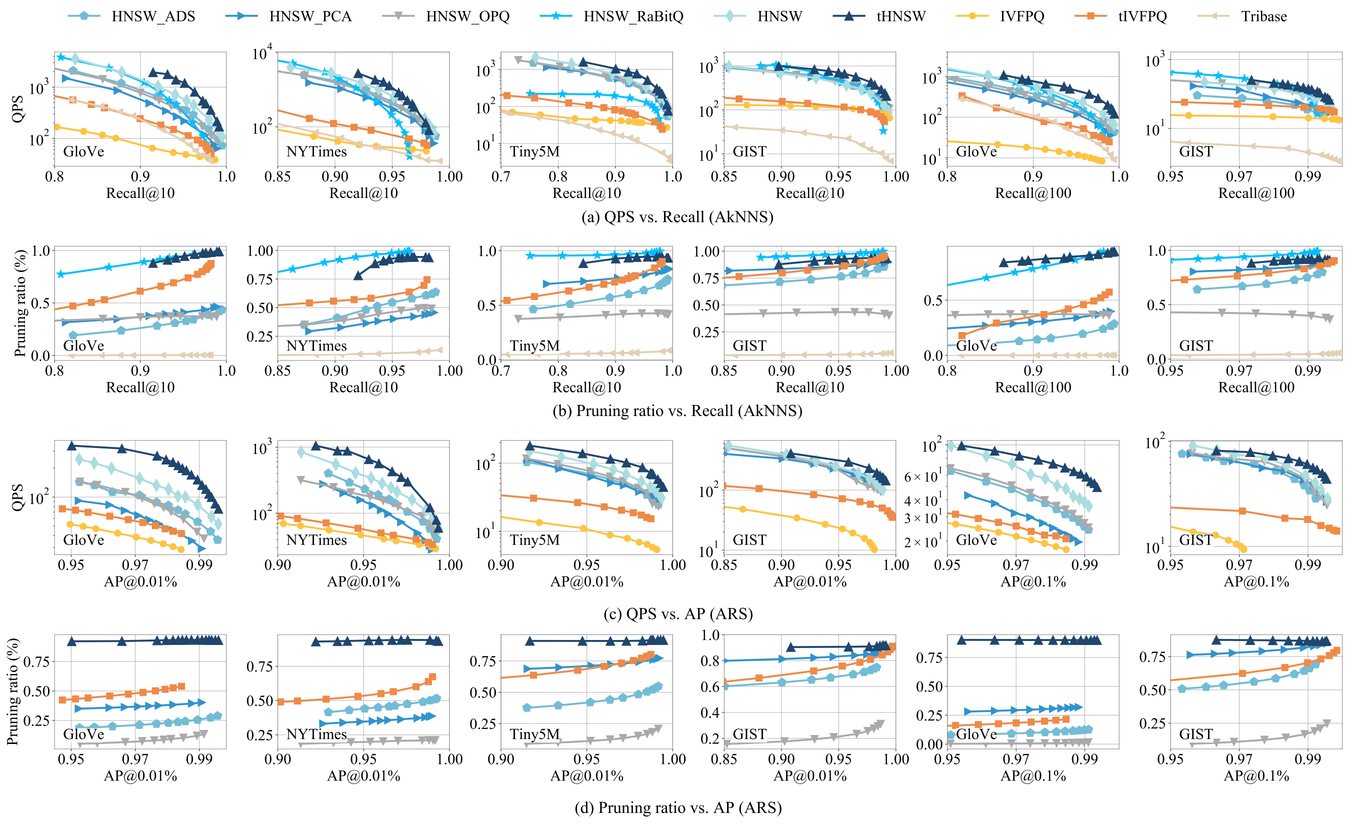


Figure 8: Overall query performance of memory-based methods

In IVFPQfs and tIVFPQfs, we use 4-bit codes and set $m = d/2$ [2]. For RaBitQ-based methods, we use 8-bit codes for full distance computation and 1-bit codes for pruning (with $m = d$ fixed and not tunable). During querying, we set $k = 10$ or 100 for AkNNS, and choose the radius for ARS such that 0.01% or 0.1% of data vectors fall within the range. The search queue size ef and accessed cluster count $nprobe$ are tuned to balance accuracy and efficiency, with the default value ensuring 0.99 recall. Other methods follow the default settings in the benchmark [57].

Implementations. All methods are implemented in C++ with SIMD optimizations ***opened***. The baselines and our method are implemented using the libraries [2, 12, 21, 58, 60] and [12], respectively. In-memory experiments are conducted on a Linux server with 512GB memory and an Intel(R) Xeon(R) Gold 6342 CPU @ 2.80GHz processor, while disk-resident experiments are performed on a machine with Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 12 vCores, 62GB memory, and a 256GB SAMSUNG NVMe SSD. We utilize all available threads to build the index and a single thread for query execution, in line with existing work [19, 54].

6.2 Comparison of Memory-Based Methods

6.2.1 HNSW-Based Methods. For AkNNS queries, Figure 8(a)–(b) depicts the trade-offs among QPS, pruning ratios, and query recall (Recall@10 and Recall@100) across different datasets. Among HNSW-based methods, not all variants outperform the original

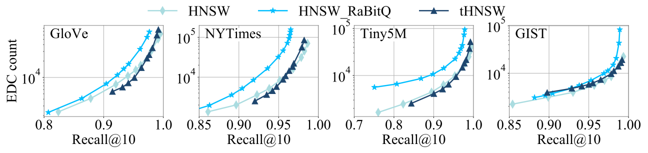
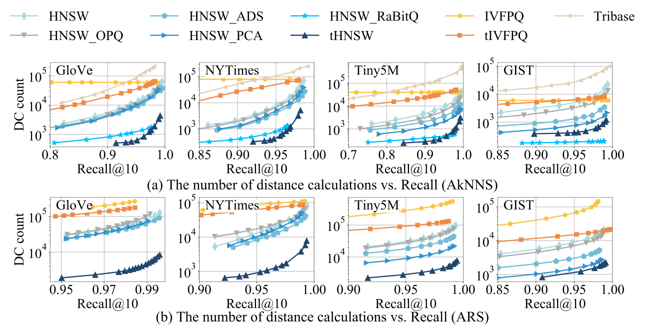


Figure 9: Comparison of estimated distance calculations

HNSW when SIMD is enabled. In contrast, tHNSW consistently achieves the best overall performance. At 99% recall for $k = 10$, it improves QPS over HNSW by 56%, 16%, 27%, and 91% on GloVe, NYTimes, Tiny5M, and GIST, respectively. For $k = 100$, it yields 97% and 88% gains on GloVe and GIST. These improvements are accompanied by high pruning ratios, up to 99.3%.

Notably, at 97% recall for $k = 10$, HNSW_RaBitQ is 77%, 87%, 74%, and 35% slower than tHNSW on GloVe, NYTimes, Tiny5M, and GIST, respectively. This is mainly because it accesses 2.6× more vectors for estimated distance calculations (EDCs), as shown in Figure 9, while tHNSW accesses a comparable number to HNSW. Moreover, we find that the 1-bit pruning in RaBitQ is often too aggressive and its 8-bit codes for full distance calculation are less accurate than exact distances, leading to high pruning ratios but reduced recall (typically ≤ 0.98).

To clarify computational cost, Figure 10(a) compares the average number of full distance calculations (DCs) versus recall. Except



(a) The number of distance calculations vs. Recall (AkNNS)

(b) The number of distance calculations vs. Recall (ARS)

Figure 10: Comparison of distance calculations

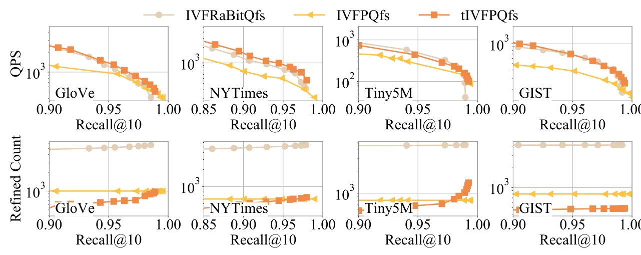


Figure 11: AkNNS performance of FastScan-based methods

on GIST, tHNSW consistently requires the fewest DCs, consistent with its superior QPS. On GIST, although HNSW_RaBitQ reduces DCs by 62% via more aggressive pruning, it incurs 54% more EDCs, resulting in comparable QPS to tHNSW.

For ARS queries, Figure 8(c)–(d) shows the QPS, pruning ratio, and accuracy trade-offs (AP@0.01% and AP@0.1%). Again, tHNSW achieves the best performance across all datasets. At 99% accuracy with $e = 0.01\%$, it outperforms the best alternative by 72%, 11%, 52%, and 53% on GloVe, NYTimes, Tiny5M, and GIST, respectively. For $e = 0.1\%$, it yields gains of 52% and 13% on GloVe and GIST. The pruning ratio remains above 99% across all datasets. Figure 10(b) further confirms that tHNSW requires the fewest DCs, reinforcing its superior efficiency.

6.2.2 IVF- or PQ-Based Methods. For AkNNS queries, tIVFPQ shows substantial improvements over the original IVFPQ. At 95% recall for $k = 10$, tIVFPQ improves by 142%, 88%, 25%, and 9% across the four datasets, respectively. For $k = 100$, the performance gains on the GloVe and GIST datasets further increase to 281% and 147%. Compared with Tribase, our method achieves over 100× higher pruning ratios, benefiting from our two key improvements on pruning. In terms of QPS, tIVFPQ outperforms Tribase by 20%, 140%, 436%, and 427% on the four datasets at 95% recall for $k = 10$.

Given that quantization-based methods are often accelerated using FastScan, we also evaluate our FastScan-compatible variant tIVFPQfs against other FastScan-based baselines, as shown in Figure 11. tIVFPQfs outperforms IVFPQfs by 27%, 50%, 15%, and 53% across the four datasets, and achieves performance comparable to IVFRaBitQfs. However, we observe that IVFRaBitQfs typically refines over 5× more candidates than tIVFPQfs. Its efficiency is largely attributed to the use of 8-bit codes for full distance calculations, which is significantly faster than the exact distance refinement used by tIVFPQfs. Nonetheless, this advantage may diminish when data access becomes expensive (e.g., disk-based storage) or when higher recall targets (≥ 0.98) are required.

For ARS queries, tIVFPQ also shows substantial gains over IVFPQ. When $e = 0.01\%$ and accuracy reaches 95%, it improves QPS by 50%, 25%, 150%, and 223% across the four datasets. At $e = 0.1\%$, the improvements on GloVe and GIST are 19% and 115%, respectively.

6.2.3 Index performance. Table 2 summarizes the build time and index size for memory-based methods. Compared to HNSW, tHNSW incurs 42%, 60%, 15%, and 18% more build time on the four datasets, primarily due to the additional PQ encoding required for landmark generation. In contrast, tIVFPQ introduces minimal overhead and maintains a build time nearly identical to IVFPQ. In terms of index size, tHNSW consumes 5%, 6%, 7%, and 6% more memory than HNSW, mainly for storing distances between data vectors and their landmarks. Similarly, tIVFPQ increases the index size by 5%, 12%, 9%, and 6% compared to IVFPQ. Notably, IVFRaBitQfs requires on average 3.7× more storage than tIVFPQ due to its use of full d-dimensional 8-bit encodings.

Table 2: Build time and memory overhead

Method	Build Time (s)				Index Size (MB)			
	GloVe	NYTimes	Tiny5M	GIST	GloVe	NYTimes	Tiny5M	GIST
HNSW	54	23	565	197	619	324	8032	3803
tHNSW	77	37	651	233	653	344	8567	4026
HNSW_ADS	57	24	767	249	619	324	8032	3803
HNSW_OPQ	230	119	2616	857	641	334	8204	3841
HNSW_PCA	56	26	810	275	619	324	8032	3803
HNSW_RaBitQ	62	21	671	224	335	118	2635	1077
IVFPQ (fs)	31	13	100	98	39	25	461	253
tIVFPQ (fs)	31	13	102	98	41	28	503	267
IVFRaBitQfs	585	357	1872	4169	176	85	1956	962
Tribase	85	49	470	457	304	77	1283	270

6.3 Comparison of Disk-Based Methods

6.3.1 AkNNS query performance. Figure 12(a) compares QPS, mean I/Os, and Recall@100 across datasets. On Cohere, tDiskANN outperforms DiskANN and Starling by 48% in QPS while reducing I/O by 25%. Similar improvements are seen on OpenAI, with 49% higher QPS and 26% fewer I/Os. The advantage is even more notable at high accuracy ($\geq$ 99% recall), where tDiskANN achieves 102% higher QPS and 58% fewer I/Os on Cohere. These gains come from reducing raw vector reads through a decoupled data layout and optimized query strategy, enabling efficient high-recall search.

6.3.2 ARS query performance. Figure 12(b) shows QPS and mean I/Os against AP@0.1%. On Cohere, tDiskANN improves QPS by 242% and 137% over DiskANN and Starling, with I/O reductions of 20% and 8%, respectively. On OpenAI, it yields 221% and 73% QPS gains with 45% and 6% fewer I/Os. Although I/O savings over Starling are modest, the QPS improvement remains substantial thanks to tDiskANN 's one-pass candidate selection, in contrast to Starling's multi-round refinement.

6.3.3 Index performance. Table 3 reports index build time and disk usage. tDiskANN has similar build time to DiskANN and Starling, with primary cost arising from PG and PQ construction. It uses

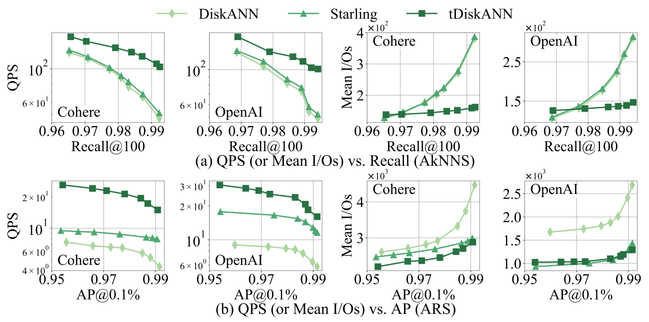


Figure 12: Overall query performance of disk-based methods

Table 3: Build time and disk overhead

Method	Build Time (s)		Disk Overhead (MB)	
	Cohere	OpenAI	Cohere	OpenAI
DiskANN	1892	1623	8572	8555
Starling	1892	1623	8572	8555
tDiskANN	1917	1633	8971	8750

slightly more storage, i.e., 5% more on Cohere and 2% on OpenAI, due to its decoupled layout. Given the large capacity of modern disks, these few extra storage overheads are perfectly acceptable.

6.4 Scalability

Figure 13(a)–(b) illustrate how the query efficiency and pruning ratio of tHNSW and tIVFPQ scale with dataset size on SIFT10M. As data size increases, QPS decreases roughly linearly. A 5× growth in data leads to only a 47% and 79% QPS drop for tHNSW and tIVFPQ, respectively. Meanwhile, the pruning ratio remains stable, demonstrating TRIM's scalability and consistent pruning effectiveness.

Figure 13(c)–(d) show the performance of tDiskANN on SIFT100M as data volume grows. Both QPS and mean I/Os scale near-linearly. With a 5× larger dataset, QPS drops just 36%, and I/Os rise by only 29%, confirming that tDiskANN maintains high throughput and low I/O overhead even at large scale.

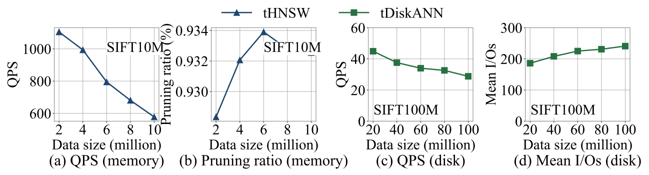


Figure 13: Scalability

6.5 Pruning Performance of TRIM

Evaluation of landmark optimization. Figure 14(a) reports the tightness of lower bounds from various landmark strategies, measured as the ratio between the lower bound and exact distance. We compare our approach with three baselines: **Random** [15, 25], which selects landmarks randomly; **Distancing** [25, 62], which maximizes inter-landmark distances; and **Clustering** [61], which uses cluster centroids as landmarks (detailed in Section 3.1). All methods apply the strict triangle inequality for lower bound calculation. On the NYTimes and GloVe datasets, TRIM achieves the highest tightness, i.e., 52% and 75% of the exact distance, respectively, surpassing baselines by up to 325%. This confirms the superiority of our landmark design in producing tighter bounds.

Evaluating p-relaxed lower bounds. Figure 14(b) compares the ratios of the strict lower bound from the triangle inequality and the p-relaxed lower bound to the exact distance, with p set to 1. On the NYTimes and GloVe datasets, the p-relaxed lower bound improves tightness by 77% and 21% over the strict lower bound, respectively. This improvement significantly enhances the pruning effectiveness of triangle-inequality-based lower bounds, enabling more effective application of triangle-inequality-based pruning in high-dimensional spaces.

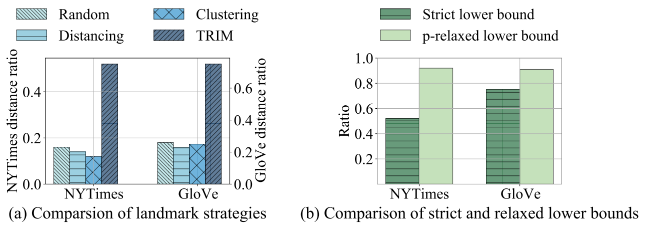


Figure 14: Evaluation of two improvements on TRIM

Ablation study. Figure 15 reports the QPS and recall when key components of TRIM are removed. For tHNSW, replacing PQ landmarks with random ones and p-LBF with strict bounds leads to average QPS drops of 56% and 64%, respectively. For tIVFPQ, we only replace p-LBF (as PQ is necessary for distance estimation), resulting in an average 63% drop across all datasets. These results show that both landmark strategy and p-relaxed lower bounds are essential and removing either severely degrades performance.

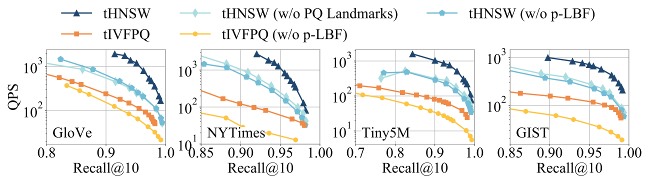


Figure 15: Ablation Study

Evaluating the impact of p on query accuracy and γ. Figures 16(a)-(b) examine how p affects γ and recall. As described in Section 3.2, increasing p lowers γ and query efficiency while improving recall. On the NYTimes dataset (with standard normal queries), $p = 0.95$ gives $\gamma = 0.85$ and recall of 0.977, while $p = 0.97$ yields $\gamma = 0.81$ and recall of 0.98. For the GloVe dataset (without explicit distribution), $p = 0.95$ results in $\gamma = 0.8$ and recall 0.97, and $p = 0.97$ lowers γ to 0.77 with recall 0.98. These results highlight the flexibility of TRIM in trading off recall and efficiency.

Evaluating the impact of γ on distance error. Figures 16(c)-(d) show how varying γ affects the gap between the p-relaxed lower bound and the exact distance. As γ increases, the lower bound becomes more aggressive (i.e., error becomes positive), increasing

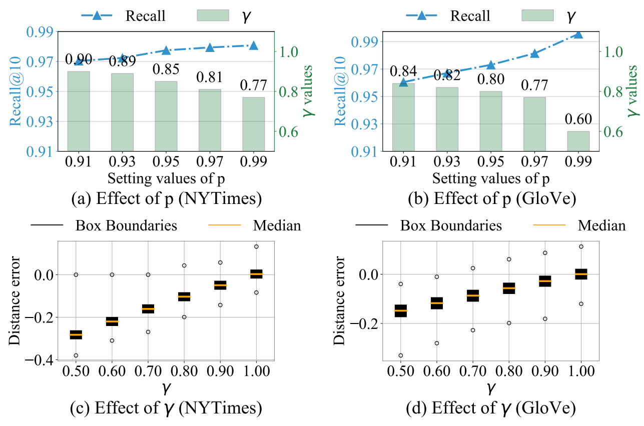


Figure 16: Evaluating the effect of p and γ

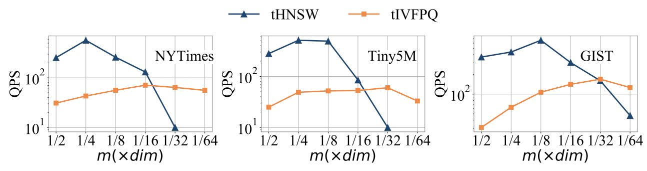


Figure 17: Evaluating the effect of parameter m

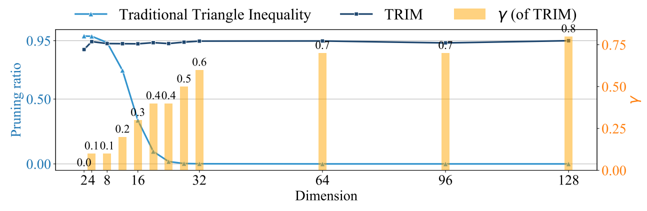


Figure 18: Comparison with traditional pruning

the risk of mistakenly pruning correct results. On NYTimes, setting $\gamma = 0.8$ causes the lower bound to slightly exceeds the true distance, which may affect recall. In contrast, using $\gamma < 0.8$ ensures zero overestimation while still tightening the bound. Similar trends are observed on GloVe. These results underscore the importance of tuning γ (i.e., p) to balance pruning aggressiveness and accuracy.

Evaluating the impact of m on TRIM. Figure 17 illustrates how QPS varies with the parameter m used for generating PQ landmarks, where m is set to 1/2, 1/4, 1/8, 1/16, 1/32, and 1/64 of the data dimensionality dim. For tHNSW, $m = dim/4$ yields the best performance on NYTimes and Tiny5M, while $dim/8$ is optimal for GIST. For tIVFPQ, query efficiency is less sensitive to m; $dim/16$ performs best on NYTimes, and $dim/32$ is optimal for both Tiny5M and GIST.

Dimensionality testing. Figure 18 compares the pruning effectiveness of TRIM with the traditional method across varying dimensionalities. Traditional methods degrade rapidly with increasing dimensionality due to the "distance concentration" phenomenon. In contrast, TRIM leverages optimized landmarks and p-LBF (with adaptive γ) to maintain both tight lower bounds and query accuracy. Empirically, the traditional method performs better when dimensionality is below 8 due to its simplicity and higher pruning ratio, but TRIM consistently outperforms it beyond this point.

7 Related Work

Triangle-inequality-based search algorithms. The triangle inequality is a key tool for enhancing query efficiency [15, 16, 25, 35, 40, 45, 62]. It has been widely applied to problems such as the shortest path query in road networks [25, 45, 62], where algorithms like ALT [25] improve the traditional A* algorithm [28] by incorporating landmarks and triangle inequalities, achieving order-of-magnitude efficiency gains. Additionally, the triangle inequality is employed to accelerate clustering algorithms, such as k-Means [16, 56] and DBSCAN [34], achieving speedups of up to 300×. In distributed systems, it also optimizes low-dimensional kNNS queries [35, 40]. However, their effectiveness diminishes in high-dimensional settings due to the "curse of dimensionality" phenomenon.

High-dimensional vector similarity search algorithms. HVSS algorithms are often classified into four categories: (1) tree-based [8, 9, 11, 13, 42], (2) hash-based [26, 29, 37, 50, 51], (3) quantization-based [4, 6, 7, 23, 24, 27, 32], and (4) proximity-graph-based [18, 19, 39, 41, 44, 64]. For in-memory storage scenarios, PG-based methods offer the best trade-off between query efficiency and accuracy [38], while quantization-based methods, particularly PQ-based approaches, excel in memory-constrained environments. In disk-resident scenarios, PQ- and PG-based methods are often combined [31, 55] to reduce I/O cost and improve search performance.

Reducing the cost of distance calculations has been one of the focuses of HVSS studies. Several distance comparison operations (DCOs) are proposed to solve this problem, which are classified into transformation-based [14, 22], projection-based [64], quantization-based [32], and geometry-based approaches [10], as summarized in the DCO benchmark [57]. Among these, transformation- and quantization-based methods offer the best overall performance in most cases [57]. Transformation-based methods accelerate calculations by dynamically adjusting the number of dimensions involved in the distance calculation. However, their dimension-level pruning often results in unnecessary data access overhead and poor SIMD compatibility, limiting their efficiency. Quantization-based methods, on the other hand, reduce computation by approximating vectors using their PQ representations for distance estimation. While efficient, they suffer from low pruning ratios due to the inherent imprecision of PQ. In contrast, our method treats PQ representations as landmarks and prunes vectors using both the fast computation of PQ and geometric principles of triangles for distance correction, achieving both high efficiency and a significantly higher pruning ratio. Tribase [61] is another triangle-inequality-based pruning method, primarily designed for clustering-based HVSS algorithms, but it faces challenges in extending to other HVSS methods, such as PG- and PQ-based approaches. Moreover, there are also several studies [36, 63] exploring early termination conditions for HVSS, which are orthogonal to ours.

8 Conclusion and Discussion

This paper studies triangle-inequality-based pruning for HVSS and highlights its limited effectiveness in high-dimensional spaces. To address this, we introduce a simple and versatile operation, TRIM, which improves pruning effectiveness through landmark optimization and p-relaxed lower bounds. TRIM can be seamlessly integrated into various widely-used HVSS solutions, including in-memory PG- and PQ-based methods, as well as disk-based approaches.

Moreover, TRIM naturally extends to other quantization methods such as RaBitQ. Specifically, the vectors reconstructed from RaBitQ's 1-bit codes can be viewed as landmarks, and the p-relaxed lower bound enhances pruning tightness and flexibility with negligible impact on accuracy, mitigating the overly aggressive and unadjustable pruning behavior of RaBitQ. Combined with RaBitQ's efficient full distance calculation using 8-bit codes, this integration holds strong potential for further performance improvements. We leave a full investigation to future work.

References

[1] Cecilia Aguerrebere, Ishwar Bhati, Mark Hildebrand, Mariano Tepper, and Ted Willke. 2023. Similarity search in the blink of an eye with compressed indices. *arXiv preprint arXiv:2304.04759* (2023).

[2] Meta AI. 2017. FAISS. https://ai.facebook.com/tools/faiss.

[3] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2016. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *VLDB*, Vol. 9. 12.

[4] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2016. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *VLDB*, Vol. 9. 12.

[5] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2019. Quicker adc: Unlocking the hidden potential of product quantization with simd. *IEEE TPAMI* 43, 5 (2019), 1666–1677.

[6] Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *CVPR*. 931–938.

[7] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE TPAMI* 37, 6 (2014), 1247–1260.

[8] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[9] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *ICML*. 97–104.

[10] 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*. 3225–3235.

[11] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, Vol. 97. 426–435.

[12] HNSWlib Contributors. 2023. HNSWlib. https://github.com/nmslib/hnswlib.

[13] Sanjoy Dasgupta and Yoav Freund. 2008. Random projection trees and low dimensional manifolds. In *STOC*. 537–546.

[14] Liwei Deng, Penghao Chen, Ximu Zeng, Tianfu Wang, Yan Zhao, and Kai Zheng. 2024. Efficient Data-aware Distance Comparison Operations for High-Dimensional Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2411.17229* (2024).

[15] Yufei Ding, Lin Ning, Hui Guan, and Xipeng Shen. 2017. Generalizations of the theory and deployment of triangular inequality for compiler-based strength reduction. In *ACM SIGPLAN*. 33–48.

[16] Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *ICML*. 147–153.

[17] Damien François, Vincent Wertz, and Michel Verleysen. 2007. The concentration of fractional distances. *IEEE TKDE* 19, 7 (2007), 873–886.

[18] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE TPAMI* 44, 8 (2021), 4139–4150.

[19] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. In *PVLDB*, Vol. 12. VLDB Endowment, 416–474.

[20] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2025. Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search. *ACM SIGMOD* 3, 3 (2025), 1–26.

[21] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2025. RaBitQ-Library. https://github.com/VectorDB-NTU/RaBitQ-Library.

[22] Jianyang Gao and Cheng Long. 2023. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *ACM SIGMOD* 1, 2 (2023), 1–27.

[23] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *ACM SIGMOD* 2, 3 (2024), 1–27.

[24] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *IEEE TPAMI*. 2946–2953.

[25] Andrew V Goldberg and Chris Harrelson. 2005. Computing the shortest path: A search meets graph theory.. In *SODA*, Vol. 5. 156–165.

[26] Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. 2020. iDEC: indexable distance estimating codes for approximate nearest neighbor search. *PVLDB* 13, 9 (2020).

[27] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *ICML*. PMLR, 3887–3896.

[28] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[29] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *PVLDB* 9, 1 (2015), 1–12.

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

[31] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).

[32] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE TPAMI* 33, 1 (2010), 117–128.

[33] K Krishna and M Narasimha Murty. 1999. Genetic K-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29, 3 (1999), 433–439.

[34] Marzena Kryszkiewicz and Piotr Lasek. 2010. TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality. In *International Conference on Rough Sets and Current Trends in Computing*. Springer, 60–69.

[35] Caitlin Kuhlman, Yizhou Yan, Lei Cao, and Elke Rundensteiner. 2017. Pivot-based distributed k-nearest neighbor mining. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017*. Springer, 843–860.

[36] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. 2020. Improving approximate nearest neighbor search through learned adaptive early termination. In *ACM SIGMOD*. 2539–2554.

[37] Jinfeng Li, Xiao Yan, Jian Zhang, An Xu, James Cheng, Jie Liu, Kelvin KW Ng, and Ti-chung Cheng. 2018. A general and efficient querying method for learning to hash. In *ACM SIGMOD*. 1333–1347.

[38] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *TKDE* 32, 8 (2019), 1475–1488.

[39] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *PVLDB* 15, 2 (2021), 246–258.

[40] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. 2012. Efficient processing of k nearest neighbor joins using mapreduce. *PVLDB* (2012).

[41] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE TPAMI* 42, 4 (2018), 824–836.

[42] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE TPAMI* 36, 11 (2014), 2227–2240.

[43] OpenAI. 2022. ChatGPT. https://chat.openai.com. Accessed: 2025-03-19.

[44] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. *ACM SIGMOD* 1, 1 (2023), 1–27.

[45] Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. 2012. Approximate shortest distance computing: A query-dependent local landmark scheme. *IEEE TKDE* 26, 1 (2012), 55–68.

[46] Milos Radovanovic, Alexandros Nanopoulos, and Mirjana Ivanovic. 2010. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research* 11, sept (2010), 2487–2531.

[47] Yong Rui, Thomas S Huang, and Shih-Fu Chang. 1999. Image retrieval: Current techniques, promising directions, and open issues. *Journal of visual communication and image representation* 10, 1 (1999), 39–62.

[48] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[49] A. Stuart and J.K. Ord. 1999. *Kendall's Advanced Theory of Statistics, Vol. 2A: Classical Inference & the Linear Model*. Oxford University Press. 865 pages.

[50] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *PVLDB* (2014).

[51] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2010. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM TODS* 35, 3 (2010), 1–46.

[52] E. Torgersen. 1972. Analysis of Noncentral Distributions. *Springer* (1972).

[53] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *ACM SIGMOD*. 2614–2627.

[54] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2022. Navigable Proximity Graph-Driven Native Hybrid Queries with Structured and Unstructured Constraints. *arXiv preprint arXiv:2203.13601* (2022).

[55] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *ACM SIGMOD* 2, 1 (2024), 1–27.

[56] Yuke Wang, Boyuan Feng, Gushu Li, Georgios Tzimpragos, Lei Deng, Yuan Xie, and Yufei Ding. 2021. TiAcc: Triangle-inequality based Hardware Accelerator for K-means on FPGAs. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 133–142.

[57] Zeyu Wang, Haoran Xiong, Zhenying He, Peng Wang, et al. 2024. Distance Comparison Operators for Approximate Nearest Neighbor Search: Exploration and Benchmark. *arXiv preprint arXiv:2403.13491* (2024).

[58] Zeyu Wang, Haoran Xiong, Zhenying He, Peng Wang, et al. 2025. Fudist. https://github.com/CaucherWang/Fudist.

[59] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. *PVLDB* 13, 12 (2020), 3152–3165.

[60] Qian Xu, Juan Yang, Feng Zhang, Junda Pan, Kang Chen, Youren Shen, Amelie Chi Zhou, and Xiaoyong Du. 2025. Tribase. https://github.com/xuqianmamba/Tribase.

[61] 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. *ACM SIGMOD* 3, 1 (2025), 1–28.

[62] Bin Yao, Mingwang Tang, and Feifei Li. 2011. Multi-approximate-keyword routing in GIS data. In *Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems*. 201–210.

[63] 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 *OSDI*.

[64] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *PVLDB* 16, 8 (2023), 1979–1991.

[65] Zihao Zhao, Zhihong Shen, and Mingjie Tang. 2021. PandaDB: Understanding Unstructured Data in Graph Database. *arXiv preprint arXiv:2107.01963* (2021).

[66] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *ACM SIGMOD* 2, 1 (2024), 1–26.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009