

Dynamically Detect and Fix Hardness for Efficient Approximate Nearest Neighbor Search

ZHIYUAN HUA, College of Computer Science, Nankai University, China

QIJI MO, College of Computer Science, Nankai University, China

ZEBIN YAO, College of Computer Science, Nankai University, China

LIXIAO CUI*, College of Computer Science, Nankai University, China

XIAOGUANG LIU*, College of Computer Science, Nankai University, China

GANG WANG, College of Computer Science, Nankai University, China

ZIJING WEI, Alibaba Group Holding Limited, China

XINYU LIU, Alibaba Group Holding Limited, China

TIANXIAO TANG, Alibaba Group Holding Limited, China

SHAOZHI LIU, Alibaba Group Holding Limited, China

LIN QU, Alibaba Group Holding Limited, China

Approximate Nearest Neighbor Search (ANNS) has become a fundamental component in many real-world applications. Among various ANNS algorithms, graph-based methods are state-of-the-art. However, ANNS often suffers from a significant drop in accuracy for certain queries, especially in Out-of-Distribution (OOD) scenarios. To address this issue, a recent approach named RoarGraph constructs a bipartite graph between the base data and historical queries to bridge the gap between two different distributions. However, it suffers from some limitations: (1) Building a bipartite graph between two distributions lacks theoretical support, resulting in the query distribution not being effectively utilized by the graph index. (2) Requires a sufficient number of historical queries before graph construction and suffers from high construction times. (3) When the query workload changes, it requires reconstruction to maintain high search accuracy.

In this paper, we first propose Escape Hardness, a metric to evaluate the quality of the graph structure around the query. Then we divide the graph search into two stages and dynamically identify and fix defective graph regions in each stage based on Escape Hardness. (1) **From the entry point to the vicinity of the query.** We propose **Reachability Fixing (RFix)**, which enhances the navigability of some key nodes. (2) **Searching within the vicinity of the query.** We propose **Neighboring Graph Defects Fixing (NGFix)** to improve graph connectivity in regions where queries are densely distributed. The results of extensive experiments show that our method outperforms other state-of-the-art methods on real-world datasets, achieving up to $2.25\times$ faster

*corresponding authors

Authors' Contact Information: Zhiyuan Hua, huazy@nbjl.nankai.edu.cn, College of Computer Science, Nankai University, Tianjin, China; Qiji Mo, moqi@nbjl.nankai.edu.cn, College of Computer Science, Nankai University, Tianjin, China; Zebin Yao, yaozb@nbjl.nankai.edu.cn, College of Computer Science, Nankai University, Tianjin, China; Lixiao Cui, cuilx@nbjl.nankai.edu.cn, College of Computer Science, Nankai University, Tianjin, China; Xiaoguang Liu, liuxg@nbjl.nankai.edu.cn, College of Computer Science, Nankai University, Tianjin, China; Gang Wang, wgzwp@nbjl.nankai.edu.cn, College of Computer Science, Nankai University, Tianjin, China; Zijing Wei, wzjr28@outlook.com, Alibaba Group Holding Limited, Beijing, China; Xinyu Liu, lxy264173@alibaba-inc.com, Alibaba Group Holding Limited, Beijing, China; Tianxiao Tang, tianxiao.ttx@alibaba-inc.com, Alibaba Group Holding Limited, Beijing, China; Shaozhi Liu, shaozhi.liu@gmail.com, Alibaba Group Holding Limited, Beijing, China; Lin Qu, xide.ql@taobao.com, Alibaba Group Holding Limited, Beijing, China.

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.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM 2836-6573/2025/12-ART318

<https://doi.org/10.1145/3769783>

search speed for OOD queries at 99% recall compared with RoarGraph and 6.88× faster speed compared with HNSW. It also accelerates index construction by 2.35-9.02× compared to RoarGraph.

CCS Concepts: • **Information systems** → **Information retrieval query processing**; **Query optimization**.

Additional Key Words and Phrases: Graph-based Index, High-Dimensional Vector, Approximate Nearest Neighbor Search

ACM Reference Format:

Zhiyuan Hua, Qiji Mo, Zebin Yao, Lixiao Cui, Xiaoguang Liu, Gang Wang, Zijing Wei, Xinyu Liu, Tianxiao Tang, Shaozhi Liu, and Lin Qu. 2025. Dynamically Detect and Fix Hardness for Efficient Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 318 (December 2025), 28 pages. <https://doi.org/10.1145/3769783>

1 INTRODUCTION

Nearest Neighbor Search (NNS) plays a crucial role in many real-world applications, such as large-scale information retrieval [2, 8, 55], recommendation systems [13, 52], and Retrieval Augmented Generation (RAG) in large language models (LLM) [18, 39, 76]. These applications require fast response times, but exact NNS in high-dimensional spaces is unable to meet the practical demands of current applications. As a result, most applications now rely on Approximate Nearest Neighbor Search (ANNS) for efficient neighbor search. To ensure high accuracy while maintaining low latency in ANNS, numerous methods have been proposed [4, 6, 7, 10–12, 19, 23–26, 28, 36, 41, 49, 50, 53, 63, 67, 71–73, 77–79]. Among these, graph-based methods currently achieve the best performance in many scenarios [23, 49, 68].

However, graph-based methods often suffer from a significant drop in accuracy for certain queries [40, 70] (referred to as ‘hard queries’ in this paper). We also observed this phenomenon in the real production environment of our e-commerce platform, which significantly affects the user experience. The main reasons for the accuracy degradation are as follows: (1) Graph-based indexes are typically approximations of Relative Neighborhood Graph (RNG) [35] or its variants [22, 23, 53], and theoretical guarantees on search accuracy only hold when the queries overlap with or are highly similar to the base data. However, in practice, not all queries are close to the base data, and queries that are farther from the base data tend to have lower accuracy. (2) Graph-based methods typically utilize a greedy search algorithm that explores only the closest vectors to the query at each step. Without effective edges in the graph index, the search may only reach a subset of the Nearest Neighbors (NNs), leading to a drop in accuracy. In particular, the drop in accuracy is more frequent in cross-modal retrieval scenarios due to the existence of the modality gap [42]. This gap causes the distributions of query vectors (e.g., texts) and base vectors (e.g., images) to differ significantly (i.e., query vectors are Out-of-Distribution (OOD)), making queries distant from the base data [11]. (The formal definition of OOD queries will be provided in Section 2.)

To improve index performance on hard queries, especially in OOD scenarios, recent methods [11, 34] aim to bridge the distribution gap between base data and queries. Among them, RoarGraph [11] is a representative method. The index construction method of RoarGraph consists of the following steps: (1) RoarGraph builds a bipartite graph between the base data X and the historical query set Q to connect the two distributions. (2) It then projects the query points onto the base data (using the nearest neighbor of query to replace the query points) to avoid adding query points directly into the graph. (3) Finally, it enhances graph connectivity by assigning more neighbors to each node. RoarGraph constructs a high-quality graph index through the above approach, which helps greedy search discover more NNs.

Despite the significant performance improvements achieved by RoarGraph, it still has the following limitations: (1) Building a bipartite graph between two distributions lacks theoretical

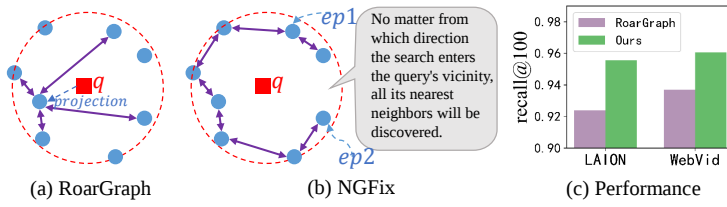


Fig. 1. Comparison between RoarGraph and Our Methods. (a) Example of RoarGraph. (b) Example of NGFix. (c) Query accuracy under same average latency in LAION10M[59] and WebVid2.5M[8].

support. As a result, some effective edges are overlooked by RoarGraph. As shown in Figure 1a, RoarGraph fails to fully utilize the information provided by historical queries, resulting in some nodes not being connected to their most effective neighbors when compared to Figure 1b. (2) It often relies on a substantial number of historical queries during the graph construction phase to achieve satisfactory performance. It also requires calculating the exact k NN for each historical query during graph construction, making it suffer from long construction times. (3) When the query workload changes, RoarGraph requires a full reconstruction to maintain accuracy. However, in real-world production environments, the hard queries are constantly changing, making it difficult for RoarGraph to adapt to such changes. For example, on our e-commerce platform, a comparison between query workloads from two different time periods shows that approximately 10% of the queries in the newer period are far from the distribution of the previous queries.

To address the limitations above, the key idea of our methods is to leverage online queries to dynamically fix defects of the graph in the regions where the queries are densely distributed. Due to the varying hardness of queries on a given graph index [40, 70], we propose **Escape Hardness (EH)**, a theoretically guaranteed method for quantifying query hardness. Following previous work [65, 70], we divide the search into two phases: (1) From the entry point to the vicinity of the query. (2) Searching within the vicinity of the query. For most queries, greedy search easily enters the second phase [70] (solution of some counterexamples will be introduced later) but often reaches only a part of the NNs. Therefore, we primarily measure the quality of the graph structure around each query. To avoid the impact of the entry point, we use a matrix to represent the difficulty of the query, where each element of the matrix indicates the difficulty of exploring from one point around the query to another point. Meanwhile, for a given query, if the values in the matrix are all small, the search accuracy will be guaranteed.

Based on EH, we propose **Reachability Fixing (RFix)** and **Neighboring Graph Defects Fixing (NGFix)** to dynamically fix defects of the graph. (1) RFix mainly focuses on the first phase of the search. For a historical query, if the search does not reach the query's vicinity, RFix first obtains the approximate nearest neighbor found by the search, and then enhances the navigation performance of the approximate nearest neighbor point by expanding its candidate neighbor set. (2) NGFix mainly focuses on the quality of the graph index in the second phase of the search. Guided by EH, it adds effective edges around the query to ensure that for each historical query, the difficulty of reaching other points from any point around the query is low. As shown in Figure 1b, NGFix ensures that greedy search finds all NNs of the query from any direction. After applying NGFix and RFix, we guarantee the search accuracy for historical queries. The results of experiments shown in Figure 1c demonstrate that our method also performs well on test queries (different from historical queries). Meanwhile, to reduce the overhead of index construction, we replace exact k NN with approximate k NN during NGFix and RFix. Our contributions are summarized as follows:

- We first provide some theoretical analysis of the relationship between query accuracy and the local graph structure around the query. We then propose Escape Hardness to measure the

difficulty of accessing other points from a given point near the query, thereby measuring the quality of the graph structure around the query. The results of experiments show that Escape Hardness is highly correlated with the actual query accuracy.

- We propose NGFix and RFix, using Escape Hardness in combination with the query distribution to dynamically detect and fix the defective areas of the graph.
- We conduct comprehensive experiments on real-world datasets, and the results demonstrate that our method significantly outperforms existing methods in both OOD and In-Distribution (ID) workloads, achieving up to $2.25\times$ faster search speed for OOD queries at a 99% recall rate and accelerating index construction by $2.35\text{-}9.02\times$. Meanwhile, compared to RoarGraph, we achieve the same search efficiency using only 8%-30% of the historical queries it requires.
- We release a new cross-modal dataset to facilitate research in ANNS, which is collected from the real-world production environment of a large-scale e-commerce platform. We also provide a well-optimized C++ library for ANN query based on our method ¹.

2 PRELIMINARIES

ANNS. In this paper, we let $X = \{x_1, x_2, \dots, x_n\} \subseteq \mathbb{R}^d$ be a finite set of n vectors in a d -dimensional space. Let $\delta(\cdot, \cdot)$ denote the distance between two points and $N_{i,q}$ denote the i -th NN of q . For a given query $q \in \mathbb{R}^d$, the objective of the Approximate k Nearest Neighbor Search (AkNNS) algorithm is to find a point set S , where S satisfies $|S| = k$, $\delta(x, q) \leq (1 + \epsilon)\delta(x', q)$ for $x \in S$, $x' \in X \setminus S$, and ϵ is a small constant satisfied $\epsilon \geq 0$. We usually use recall and relative distance error (rderr) to evaluate the accuracy of query q . For a query q , $\text{recall}@k = |S \cap KNN(q)|/k$, where $KNN(q)$ represents the exact k NN of the query q ; $\text{rderr}@k = \text{avg}_{i=1}^k (\delta(ANN_{i,q}, q) / \delta(N_{i,q}, q) - 1)$, where $ANN_{i,q}$ denotes the i -th approximate NN of q .

Algorithm 1 : Greedy Search

Input : Graph index G , query q , result size k , entry point ep , search list size $L \geq k$.

Output : k NN of q

```

1 Initialize a candidate set  $C \leftarrow \{ep\}$  and a result set  $R \leftarrow \{ep\}$ 
2 mark  $ep$  as visited
3 while  $C \neq \emptyset$  do
4    $u \leftarrow \text{pop an element from } C \text{ with minimum } \delta(u, q)$ 
5    $d_{\max} \leftarrow \max_{x \in R} \delta(x, q)$  // If  $R = \emptyset$ ,  $d_{\max} \leftarrow \text{inf}$ 
6   if  $\delta(u, q) > d_{\max}$  then break
7   for  $v \in \text{Neighbors}(G, u)$  and  $v$  has not been visited do
8     mark  $v$  as visited
9     if  $\delta(v, q) < d_{\max}$  or  $|R| < L$  then
10       $R \leftarrow R \cup \{v\}$ ,  $C \leftarrow C \cup \{v\}$ 
11     while  $|R| > L$  do
12       pop an element  $x$  from  $R$  with maximum  $\delta(x, q)$ 
13      $d_{\max} \leftarrow \max_{x \in R} \delta(x, q)$ 
14 return closest  $k$  points in  $R$ 

```

Graph Index. We use a directed graph $G = (V, E)$ to represent a graph index, where V is the set of vertices, and each vector in X corresponds to a vertex in V . The set E represents the edges, where each edge $(u, v) \in E$ indicates a proximity relationship between vertices u and v . We let $\text{Neighbors}(G, v)$ denote the set of neighbors of vertex v in graph G .

Greedy Search. We typically use Greedy Search for ANNS on graph indexes. The details of the algorithm are presented in Algorithm 1. We begin by initializing a candidate set C and a result set

¹<https://github.com/yuhuifishash/NGFix>

R , where R has a maximum length of L (line 1). During the search, at each step, we select the point from C that is closest to the query q and check its neighbors. If any neighbor is closer to the query than a point in R , we add it to both R and C (lines 7-13). The search terminates when the distance from the current point to q is greater than that of any point in R or when the C becomes empty (lines 3-6). Finally, the algorithm returns the k closest points to the q from R . Since this method is an approximate algorithm, Greedy Search often terminates after exploring only a portion of the query's nearest neighbors for certain queries, leading to reduced search accuracy. A common and simple approach to address this is to increase the search list size L , but this will lead to an increase in search latency. A more effective solution is to design an improved graph index that guides the greedy search more effectively, which is the main focus of this paper.

OOD Queries. In cross-modal ANNS, the base and query vectors are typically generated by a multimodal model (e.g., CLIP [56]). Taking image-text retrieval as an example, image and text embeddings are generated by different encoders, and then these embeddings are mapped into a shared space through contrastive learning [56]. However, there exists a modality gap between image and text vectors, so their distributions are different. A query is considered OOD if its modality differs from that of the base data. We can use the Wasserstein distance [37] to mathematically measure the difference between the two distributions and use the Mahalanobis distance [47] to measure the distance from a vector to a distribution [11].

3 RELATED WORK

The algorithm of ANNS can be broadly categorized into tree-based [4, 41, 66], hashing-based [14, 63, 72, 79], quantization-based [3, 24, 26, 27, 29, 36, 50], and graph-based methods [11, 23, 31, 48, 49, 61]. Tree-based methods perform well in low-dimensional datasets, but their search performance often degrades in high-dimensional spaces due to the curse of dimensionality. Hashing-based methods provide theoretical guarantees on search results, but in practice, it is difficult to maintain high search efficiency while achieving high accuracy. Quantization-based methods compress high-dimensional vectors into short quantization codes, significantly reducing the cost of distance computation. Graph-based methods achieve the best time-accuracy trade-off across various scenarios. Moreover, they can be combined with other methods to achieve better overall performance [21, 28, 45, 78]. Since our method focuses on optimizing graph-based indexes, we next provide an overview of existing graph-based techniques.

Most state-of-the-art graph indexes are based on the Delaunay Graph [15] (DG) and the Relative Neighborhood Graph (RNG) [35]. DG is the dual graph of the Voronoi diagram [5]. In a DG, two vertices are connected by an edge if and only if there exists a circle passing through these two points, with no other points inside the circle. DG guarantees that for any given query q , a Greedy Search will be able to find $N_{1,q}$ [49]. However, DG becomes a complete graph in high-dimensional space [30], resulting in search performance similar to brute-force search. RNG is a subgraph of DG. If an edge is the longest edge of any triangle formed by vertices in base data X , that edge is not included in the RNG. RNG guarantees that the average degree of each point is a small constant when points in X are uniformly distributed, but it does not ensure the accuracy of the search. Later, Fu et al. [23] introduced the Monotonic Relative Neighborhood Graph (MRNG). MRNG preserves the property of constant average degree in RNG and guarantees that if query $q \in X$, a Greedy Search will find $N_{1,q}$ definitely. However, there is no theoretical guarantee for queries where $q \notin X$. Subsequently, Fu et al. [22] extended MRNG to SSG. It provides some probabilistic guarantees for queries q where $q \notin X$ under random distributions. Recently, Peng et al. [53] introduced τ -MG, which guarantees that if $\delta(N_{1,q}, q) < \tau$, Greedy Search can find $N_{1,q}$. Due to the high graph construction overhead of the graph mentioned above, which makes them hard to use in practice, most state-of-the-art graph indexes are approximations of these algorithms. HNSW [49] uses the

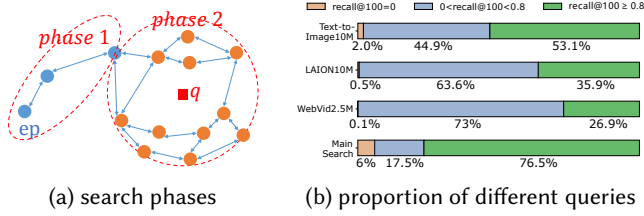


Fig. 2. (a) The two phases of Greedy Search on the graph. (b) The proportion of queries with different recall levels.

RNG rule to sparsify the graph. It also uses a hierarchical structure, but the effect of this structure is limited in high-dimensional space [43, 69]. NSG, NSSG, and τ -MNG are approximations of MRNG, SSG, and τ -MG, respectively. They primarily consider the properties of these graphs in a local neighborhood around each point, thereby reducing the indexing construction overhead. However, these indexes tend to perform poorly when the query is far from the base data (e.g., certain ID queries or OOD queries).

To overcome the issues of hard queries, especially OOD queries, RobustVamana [34] adds query vectors from other distributions into the graph and connects edges between the query and base points, where these query points serve as navigators. This partially mitigates the accuracy loss caused by OOD queries. However, these points also extend the search path, leading to only a small overall improvement. Chen et al. [11] proposed RoarGraph, which establishes a connection between the two distributions and avoids directly inserting query points into the graph by projecting them onto the base data. This approach achieves significantly better performance than existing graph indexes.

There are also some other algorithms [10, 12, 19, 25, 40, 45, 46, 54, 57] that optimize the graph search process, such as iQAN [54], which improves the performance and accuracy of ANNS by intra-query parallelism. LSH-APG [78] and some other works [45, 57] focus on solving the entry point selection problem. Additionally, some approaches [12, 16, 25] aim to optimize the distance calculations during the search process.

4 ANALYSIS

In this section, we mainly discuss the theoretical foundation for constructing graphs based on query distributions, along with preliminary experiments that support our design.

To provide theoretical support, we first explore the relationship between the graph structure and accuracy for a given query. Following previous studies [65, 70], we divide the search into two phases: (1) searching from the entry point to the vicinity of the query, and (2) searching within the vicinity of the query. Greedy Search can easily reach the second phase, but it often ends up retrieving only a subset of the nearest neighbors due to the lack of effective edges in the graph index [70] (Figure 2a illustrates an example of two search phases). Our experiments demonstrate that this conclusion holds for OOD queries as well. Figure 2b shows the proportion of queries with different recall levels when using HNSW as the graph index in four cross-modal datasets (the search list size is fixed at 100). For most queries, Greedy Search successfully reaches the query vicinity (i.e., $\text{recall}@100 \neq 0$). Therefore, we first focus on the case where the entry point is close to the query (Other cases will be discussed in Section 5.4). Subsequently, since Greedy Search only explores points closest to the query, we only need to consider the graph structure in the small region around the query. To better describe the area around the query, we first define the graph structure around a query [65], which consists of the query's NNs and the edges associated with them:

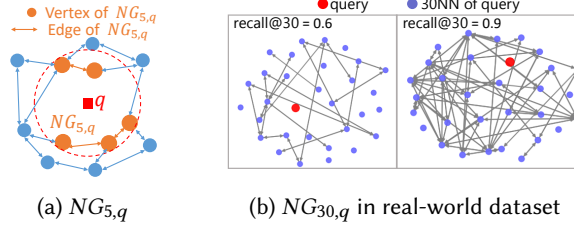


Fig. 3. (a) A toy example of $NG_{5,q}$. (b) $NG_{30,q}$ of HNSW's base layer with two different OOD queries in LAION10M.

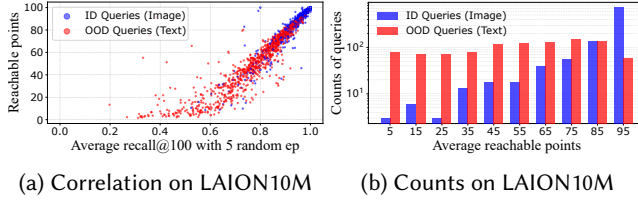


Fig. 4. Randomly sampled 1000 ID queries and 1000 OOD queries in LAION10M. The graph index is the base layer of HNSW. (a) The correlation between query's recall@100 with a search list size of 100 and its average number of reachable points in $NG_{100,q}$. (b) Counts of queries with different average numbers of reachable points in $NG_{100,q}$.

DEFINITION 1. (*S-Neighboring Graph with query q ($NG_{S,q}$)*). Consider a directed graph index $G = (V, E)$ and a query q , and define $V_{S,q} = \{N_{1,q}, N_{2,q}, \dots, N_{S,q}\}$. Then $NG_{S,q} = (V_{S,q}, E_{S,q})$ is the subgraph induced in G by $V_{S,q}$, where $E_{S,q}$ is the edge set of the subgraph.

Figure 3a shows an example of $NG_{5,q}$, where the orange vertices form $V_{5,q}$, and the orange edges form $E_{5,q}$. Figure 3b illustrates $NG_{30,q}$ for two OOD queries on the LAION10M [59] dataset, using an index constructed with HNSW's base layer. To facilitate visualization, we applied Multidimensional Scaling (MDS) [64] to project the high-dimensional data to two dimensions. The recall@30 shown in the figure represents the results of Greedy Search with a randomly selected entry point and a search list size of 30. It can be observed that for queries with poor accuracy, $NG_{30,q}$ exhibits weaker connectivity and contains numerous isolated points.

We then examine the relationship between $NG_{S,q}$ and query accuracy from two perspectives: (1) A high-quality $NG_{S,q}$ guarantees high query accuracy; (2) A low-quality $NG_{S,q}$ can easily lead to a significant drop in accuracy. **For the first perspective**, we introduce the following theorem to show that if base points u and v are connected in $NG_{S,q}$, then Greedy Search starting from u is guaranteed to reach v when the search list size $L \geq S$. All the proof in this paper and more details can be found in Appendix B ².

THEOREM 1. For a directed graph index $G = (V, E)$ and a query q . If $N_{i,q}$ can reach $N_{j,q}$ in $NG_{S,q}$ ($1 \leq i, j \leq S$). The Algorithm 1 will always visit $N_{j,q}$ when query = q , $ep = N_{i,q}$ and $L \geq S$.

Theorem 1 demonstrates that a $NG_{S,q}$ with strong connectivity can guarantee search accuracy. Furthermore, **for the second perspective**, our experiments show that poor connectivity in the $NG_{S,q}$ significantly degrades search accuracy. We measure the connectivity of $NG_{S,q}$ by calculating the average number of points that can be reached starting from a randomly chosen point in $NG_{S,q}$. Figure 4a shows the correlation between query accuracy and the average number of reachable

²<https://github.com/yuhuihash/NGFix/blob/main/appendix.pdf>

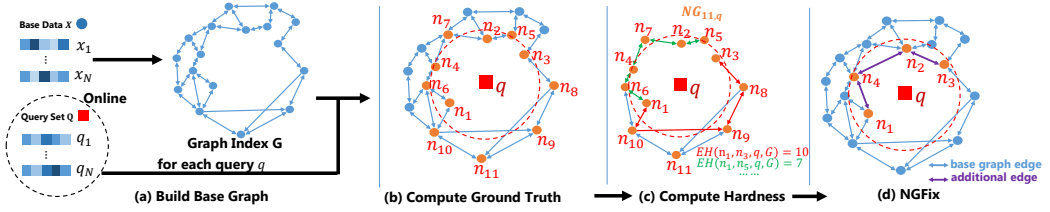


Fig. 5. An overview of Neighboring Graph Defects Fixing with $N_q = 4, K_h = 4$.

points in $NG_{100,q}$, demonstrating that insufficient connectivity in the $NG_{S,q}$ can lead to a substantial drop in search accuracy. Moreover, it can be observed that a small number of queries have a strongly connected $NG_{S,q}$, but their average recall@100 is still below 1. This is because for these queries, if the search starts from certain entry points, it may fail to reach the neighborhood of the query (i.e., fail to enter the second phase of the search).

Connectivity of $NG_{S,q}$ in Real-World Datasets. Figure 4b shows the connectivity of $NG_{100,q}$ in LAION10M. We observe that the connectivity of their $NG_{100,q}$ varies significantly for both ID and OOD queries. Although the connectivity of $NG_{100,q}$ for OOD queries is poorer compared to that of ID queries in general, there are still some special cases: approximately 30% of OOD queries have a highly connected $NG_{100,q}$, while about 10% of ID queries have an $NG_{100,q}$ with very few edges. This means that when using queries to guide graph construction, we should focus on the hard queries with low-quality $NG_{S,q}$.

Reason for Using Historical Queries. Although the graph structure around the query impacts the query accuracy, the following theorem we propose demonstrates that for any query and S , ensuring strong connectivity in $NG_{S,q}$ is impractical.

THEOREM 2. *Given a base data set X and the DG constructed from the points in X . If any edge of DG is removed, there exists a query q such that $NG_{2,q}$ contains only two isolated nodes.*

Since DG becomes a complete graph in high-dimensional spaces [30], ensuring strong connectivity for the $NG_{S,q}$ of all queries is impractical in the real world. Therefore, the method we propose below leverages the historical queries to achieve strong connectivity for the $NG_{S,q}$ in regions where queries are densely distributed.

5 DESIGN

The analysis in Section 4 can be summarized as follows: (1) The difficulty of a query is largely influenced by the quality of the graph structure around it, and the difficulty varies significantly across queries. So we first propose EH to measure the quality of the graph structure around a query in this section. (2) We need to utilize historical queries to enhance graph connectivity in regions where queries are densely distributed. So we then propose NGFix, using EH in combination with historical queries to guide us in fixing the defective areas of the graph. Figure 5 illustrates the overview of NGFix. We begin by constructing a base graph (Figure 5a). For each query q , we compute its exact KNN (Figure 5b) and then calculate EH for q (Figure 5c). Finally, we add additional edges in the vicinity of q to fix the defective regions of the graph (Figure 5d). (3) There still exist a few queries for which the search stops in the first phase and cannot reach their vicinity. So we propose RFix to enhance the navigability of some points.

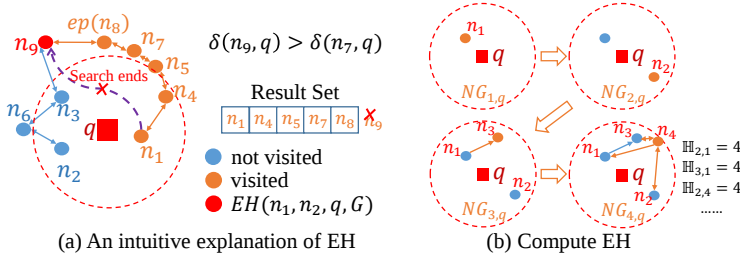


Fig. 6. Example of Escape Hardness.

5.1 Preprocess

First, we need to construct any type of graph index as a base graph, such as HNSW, NSG, etc. Then, for each query, we need to compute its exact k NN. We have the following methods for calculating k NN:

- Collect the queries and process them after accumulating a certain amount (e.g., 512), and then transform the computation of exact k NN into matrix multiplication, and accelerate it using matrix multiplication optimizations [17].
- Utilize the previously constructed graph index to perform Greedy Search, but increase the search list size to improve the accuracy of approximate k NN. This method is significantly faster than brute-force computation, but may slightly affect the ground truth quality of the queries. The results of experiments in Section 6.4 demonstrate that this method has little impact on the final graph's quality.

5.2 Escape Hardness

To better measure the quality of the graph structure around a query, we introduce Escape Hardness. It focuses on measuring the difficulty of reaching one point from another around the query using Greedy Search.

DEFINITION 2. (*Escape Hardness (EH)*). Given a directed graph index $G = (V, E)$ and a query q , let $P(u, v, G)$ denotes the set of all paths from vertex u to vertex v in graph G and $F_{u,q}$ denote the point u is the $F_{u,q}$ -th NN of q , the Escape Hardness of query q from vertex u to vertex v is defined as:

$$EH(u, v, q, G) = \min_{p \in P(u, v, G)} \max_{x \in p} F_{x,q}$$

and we define the Escape Hardness Matrix $\mathbb{H} \in \{1, 2, \dots, |X|\}^{N_q \times N_q}$ where $\mathbb{H}_{i,j} = EH(N_{i,q}, N_{j,q}, q, G)$ and N_q represents the number of nearest neighbors considered for query q .

The definition of $EH(u, v, q, G)$ illustrates the hardness of searching from a point u to reach v using Greedy Search. The validity of EH is primarily supported by the following corollary of Theorem 1. EH establishes an upper bound on the search list size required to successfully reach a point from another point around the query.

COROLLARY 1. For a directed graph index $G = (V, E)$ and a query q , the Algorithm 1 will always visit $N_{j,q}$ when query = q , $ep = N_{i,q}$ and $L \geq EH(N_{i,q}, N_{j,q}, q, G)$.

Example. Figure 5c is an example for EH: For simplicity, we use n_i to represent $N_{i,q}$ in the figure. For $EH(n_1, n_3, q, G)$, without considering the detour through n_6 , there are mainly two paths from n_1 to n_3 : $p_1 = \{n_1, n_{10}, n_9, n_8, n_3\}$ and $\max_{x \in p_1} F_{x,q} = 10$, $p_2 = \{n_1, n_{10}, n_{11}, n_9, n_8, n_3\}$ and $\max_{x \in p_2} F_{x,q} = 11$. Since $\max_{x \in p_1} F_{x,q}$ is smaller, $EH(n_1, n_3, q, G) = 10$, and $\mathbb{H}_{1,3} = 10$. **Intuition.** In Figure 6a, we take n_8 as the entry point and aim to recall the top-2 NN of q using a result set of size 5

(i.e., $L = 5$ in Algorithm 1). After the search visits n_1 , the result set becomes full. Since n_9 is far from the query, it will be removed from the result set. Then the greedy search converges prematurely. It will not go back to n_9 and expand its neighbors, resulting in $\text{recall}@2 = 50\%$. Here, EH measures the difficulty of reaching n_2 from n_1 along the search path. Specifically, $\text{EH}(n_1, n_2, q, G) = 9$, meaning the critical point (n_9) hindering the search ranks 9th in terms of distance to q . In this example, increasing L to 6 would allow the search to reach n_2 . However, it is generally difficult to determine the exact L needed to ensure a certain point is visited, so EH provides an upper bound on L .

To compute EH, we first present the following theorem:

THEOREM 3. *If points $N_{i,q}$ and $N_{j,q}$ satisfy the condition that $N_{i,q}$ cannot reach $N_{j,q}$ in $NG_{S-1,q}$ but can reach $N_{j,q}$ in $NG_{S,q}$, then $\text{EH}(N_{i,q}, N_{j,q}, q, G) = S$ and $\mathbb{H}_{i,j} = S$.*

Algorithm 2 : Compute Hardness

Input : Query q , $NG_{MaxS,q} = (V_{MaxS,q}, E_{MaxS,q})$, N_q

Output : The Escape Hardness Matrix $\mathbb{H}$

```

1 Initialize a matrix  $f \in \{0, 1\}^{MaxS \times MaxS}$  with all elements set to 0
2 Set all elements of  $\mathbb{H}$  to  $\infty$ 
3 Let  $f_{i,j} \leftarrow 1$  if  $(i, j) \in E_{MaxS,q}$  or  $i = j$  for all  $1 \leq i, j \leq N_q$ 
4 Let  $\mathbb{H}_{i,j} \leftarrow \max(i, j)$  if  $f_{i,j} = 1$  for all  $1 \leq i, j \leq N_q$ 
5 for  $h \leftarrow 1$  to  $MaxS$  do                                     // add points to subgraph.
6     for  $i \leftarrow 1$  to  $MaxS$  do
7          $C \leftarrow \emptyset$                                      // Save the connectivity-changed points.
8         for  $j \leftarrow 1$  to  $MaxS$  do                             // Floyd-Algorithm.
9              $f_{i,j} \leftarrow f_{i,j}$  or  $(f_{i,h}$  and  $f_{h,j})$ 
10            if  $f_{i,j}$  changed then  $C \leftarrow C \cup \{j\}$ 
11        if  $i \leq N_q$  then
12            for  $u \in C$  and  $u \leq N_q$  do
13                 $\mathbb{H}_{i,u} = \max(i, u, h)$ 
14 return  $\mathbb{H}$ 
  
```

Due to THEOREM 3, when computing EH, we start with $S = 1$ and incrementally consider $NG_{S,q}$ in ascending order of S . For the subgraph $NG_{S-1,q}$, we add the point $N_{S,q}$ and its associated edges to construct $NG_{S,q}$. If the connectivity from points $N_{i,q}$ to $N_{j,q}$ changes after adding point $N_{S,q}$, then $\mathbb{H}_{i,j} = S$. The connectivity updates after each addition are computed using a modified Floyd-Warshall algorithm [20]. The calculation of EH is finished when $N_{i,q}$ and $N_{j,q}$ are reachable from each other for any $1 \leq i, j \leq N_q$. We denote the value of S at the time the computation is completed as $MaxS$, also meaning that the subgraph at this point is $NG_{MaxS,q}$. However, based on the previous analysis, the connectivity of the graph around hard queries tends to be poor, often resulting in a significantly large value of $MaxS$. Therefore, to efficiently compute EH, we limit $MaxS$ to a constant multiple of N_q (e.g., $MaxS \leq 5N_q$). If $N_{i,q}$ and $N_{j,q}$ are still not connected at this point, we consider $\mathbb{H}_{i,j}$ to be infinite. Figure 6b shows an example of computing EH: After adding n_1 and n_2 , none of the nodes can reach each other, so we cannot compute any EH now. When n_3 is added, n_1 becomes able to reach n_3 , so we set $\mathbb{H}_{1,3} = 3$. After adding n_4 , n_2 can reach n_1 through n_4 , so $\mathbb{H}_{2,1} = 4$. For other node pairs that also become reachable via n_4 , their EH values are also set to 4.

The details of the algorithm are shown in Algorithm 2. To accelerate the computation, we directly set $MaxS$ to $5N_q$ and preprocess $NG_{MaxS,q}$ before calculation. The matrix f represents the transitive closure, and $f_{i,j}$ indicates whether $N_{i,q}$ can reach $N_{j,q}$ (line 1). Then we set the initial values of f and $\mathbb{H}$ based on the edges in $NG_{MaxS,q}$ (lines 2-4). The loop in line 5 adds a new point in each

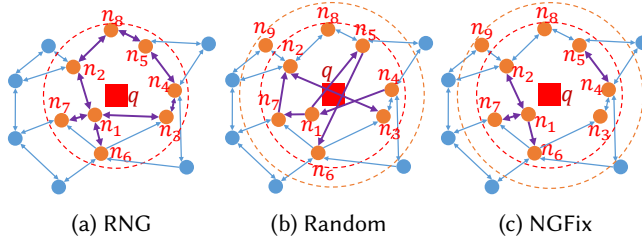


Fig. 7. The results of different methods for fixing the $NG_{8,q}$, where blue edges represent the edges of the base graph and purple edges represent the additional edges we add. (a) Reconstructing the RNG graph in the $NG_{8,q}$. (b) Randomly adding edges until any two points in the $NG_{8,q}$ are K_h -reachable ($K_h = 9$). (c) NGFix with $N_q = 8$ and $K_h = 9$.

iteration. After adding points, we perform the Floyd algorithm to compute the connectivity of the new graph and store the points where connectivity changes (lines 6-10). Finally, we update the matrix $\mathbb{H}$ based on the changes (lines 11-13). In our implementation, since the matrix f is a boolean matrix, we use bitset to store f and speed up the Floyd algorithm.

Comparison to Steiner-hardness [70]. Recently, *Steiner-hardness* effectively estimates the query difficulty with high accuracy. However, its focus differs from that of EH: (1) EH provides a finer-grained measure of the difficulty of reaching each point from any other in a query (represented as a matrix), while *Steiner-hardness* gives an overall difficulty score (single value). (2) *Steiner Hardness* focuses on estimating the computational cost required to achieve a target accuracy (e.g., recall@100 = 0.9), whereas EH aims to provide theoretical support for graph construction by measuring the upper bound of the search list size required for greedy search to reach a target from a starting point.

5.3 Neighboring Graph Defects Fixing

Using Escape Hardness to guide graph construction. Since Escape Hardness describes the hardness of Greedy Search traversing from one point to another around a query, our key idea is to **ensure that the Escape Hardness between all point pairs in the small region surrounding each historical query is low** (i.e., fix the defects in the graph). We first define a threshold K_h ($K_h \geq N_q$) to determine whether it is hard to travel from one point to another.

DEFINITION 3. (K_h -reachable). Given a directed graph index $G = (V, E)$ and a query q , if point u and v satisfy $EH(u, v, q, G) \leq K_h$, we consider u to be K_h -reachable to v . Then we define the K_h -reachable Matrix $\mathbb{T} \in \{0, 1\}^{N_q \times N_q}$ as: $\mathbb{T}_{i,j} = \begin{cases} 1 & \mathbb{H}_{i,j} \leq K_h \\ 0 & \mathbb{H}_{i,j} > K_h \end{cases}$

According to Corollary 1, ensuring low Escape Hardness between point pairs guarantees the accuracy of Greedy Search. Therefore, our next step is to develop an effective method to fix the defects in the graph.

Simple Solutions and Limitations. (1) Reconstruct RNG within the $NG_{S,q}$ for each historical query q and overlay it with the original graph. Figure 7a illustrates an example of constructing the RNG graph. This method provides high-quality neighbors for each point in $NG_{S,q}$. However, it connects many edges for individual queries, which increases the computational burden during search and enlarges the memory footprint of the graph index. (2) Randomly selecting points $N_{i,q}$ and $N_{j,q}$ with $\mathbb{T}_{ij} = 0$ in $NG_{S,q}$ and adding edges until each point is K_h -reachable to any other points in $NG_{S,q}$ (Figure 7b). However, this method results in disordered connections, and the points in $NG_{S,q}$ are not connected to their actual neighbors.

Algorithm 3 : Neighboring Graph Defects Fixing**Input** : Query q , Graph Index $G = (V, E, E_{EX})$, N_q , degree limitation of extra edges $M_{EX}, \mathbb{T}, \mathbb{H}$ **Output** : New Graph Index G_{EX}

```

1  $EC \leftarrow \emptyset$  // Edge Candidates
2 for  $(i, j) \in \{(i, j) | \mathbb{T}_{i,j} = 0, 1 \leq i, j \leq N_q\}$  do
3    $EC \leftarrow EC \cup \{(i, j)\}$ 
4 Sort  $(i, j) \in EC$  in ascending order by  $\delta(N_{i,q}, N_{j,q})$ 
5 while  $EC \neq \emptyset$  do
6    $(s, t) \leftarrow \text{pop the first element of } EC$ 
7   if  $\mathbb{T}_{s,t} \neq 0$  then continue;
8    $\mathbb{T}_{s,t} \leftarrow 1$ 
9   if  $|\text{ExtraNeighbors}(N_{s,q})| < M_{EX}$  then // Link new edge
10     $E_{EX} \leftarrow E_{EX} \cup \{(N_{s,q}, N_{t,q}, \mathbb{H}_{s,t})\}$ 
11  else // Prune Edges
12    choose  $(N_{s,q}, v, h) \in E_{EX}$  with minimum  $h$ 
13    if  $h < \mathbb{H}_{s,t}$  then
14       $E_{EX} \leftarrow E_{EX} \setminus \{(N_{s,q}, v, h)\}$ 
15       $E_{EX} \leftarrow E_{EX} \cup \{(N_{s,q}, N_{t,q}, \mathbb{H}_{s,t})\}$ 
16  for  $i \leftarrow 1$  to  $N_q$  do
17    for  $j \leftarrow 1$  to  $N_q$  do
18       $\mathbb{T}_{i,j} \leftarrow \mathbb{T}_{i,j}$  or  $(\mathbb{T}_{i,s}$  and  $\mathbb{T}_{t,j})$  // update  $\mathbb{T}$ .
19 return  $G_{EX} = (V, E, E_{EX})$ 

```

NGFix. After considering both the graph's neighborhood properties and the number of additional edges, we choose to use the idea of the minimum spanning tree (MST) [1], and MST is a subset of RNG [35]. We consider each edge $(N_{i,q}, N_{j,q})$ in increasing order of weight $\delta(N_{i,q}, N_{j,q})$. If $\mathbb{T}_{i,j} = 0$, we add that edge to the graph. Additionally, adding a new edge will make some points K_h -reachable to others, so we update $\mathbb{T}$ after each edge is added to the graph. This process is repeated until all elements of $\mathbb{T}$ are equal to 1. Figure 7c and 5d show the final result of NGFix in a toy example. The details of the algorithm are shown in Algorithm 3. We represent a graph index as $G = (V, E, E_{EX})$, where E denotes the edges of the base graph, and E_{EX} represents the additional edges we add. The main part of the algorithm is a loop that considers each edge in ascending order of distance. In each iteration, we first check whether the edge needs to be added (lines 6-8). Then, we set an additional out-degree limit M_{EX} to prevent any point from having too many extra edges (lines 10-11). When a point exceeds this limit, we prioritize pruning the edges with lower EH (lines 13-16). Finally, we update $\mathbb{T}$. The key idea is that for all $1 \leq i, j \leq N_q$, if $N_{i,q}$ is K_h -reachable to $N_{s,q}$ and $N_{t,q}$ is K_h -reachable to $N_{j,q}$, then after connecting $N_{s,q}$ and $N_{t,q}$, $N_{i,q}$ will become K_h -reachable to $N_{j,q}$ (lines 17-19).

Analysis. The following theorem provides the maximum number of edges that NGFix may add for a given query.

THEOREM 4. *Given a query q , Algorithm 3 will add at most $2(N_q - 1)$ extra directed edges.*

Based on Theorem 4, the loop in Algorithm 3 will execute at most $O(N_q)$ times before terminating. Each iteration of the loop needs to update $\mathbb{T}$, which takes $O(N_q^2)$ time, so the time complexity of Algorithm 3 is $O(N_q^3)$. For the calculation of EH (Algorithm 2), we need to add the $MaxS$ nearest neighbors of the query sequentially, and each addition requires $O(MaxS^2)$ time to compute connectivity. Therefore, the time complexity of EH calculation is $O(MaxS^3)$. Since we limit $MaxS$ to at most a constant multiple of N_q , the time complexity of our algorithm for a single query is $O(P + N_q^3)$, where P represents the time complexity of preprocessing (Section 5.1). And each query

will at most increase the size of the graph index by $O(N_q)$. Since N_q is generally set to a small value (e.g., 100), the main time overhead of our algorithm lies in the preprocessing step (i.e., computing the ground truth of each historical query).

5.4 Reachability Fixing

Previously, we focused on cases where the search successfully reaches the vicinity of the query. However, for a small subset of queries, Greedy Search fails to do so and stops before approaching the query. In the following, we discuss how to address situations where the search terminates far from the query. When this occurs, it indicates that the current point lacks outgoing edges leading toward the query. Recall that most existing graph index construction strategies approximate the RNG or its variants [11, 22, 23, 49, 53]. Their approximation process typically involves two steps: (1) For a given point, treat it as a query and use Greedy Search to retrieve nearby points, typically with a result set size ranging from 100 to 2000. (2) Apply an edge pruning strategy (e.g., MRNG) to the result set to select the true neighbors. However, since candidate neighbors are selected from a relatively small greedy search result set, the retrieved points may be clustered in a single direction. As a result, valuable neighbors in other directions may be overlooked. If the query lies in one of these neglected directions, the Greedy Search is likely to stop early without finding closer points to the query.

According to the analysis above, we propose Reachability Fixing (RFix). The key idea of RFix is to **expand the pruning candidate set of the $ANN_{1,q}$'s neighbors for searches that fail to reach the vicinity of the historical query**, where $ANN_{1,q}$ refers to the approximate nearest neighbor returned by the Greedy Search. The details are shown in Algorithm 4 and the steps are as follows: (1) **Select the centroid of the base data as the entry point**. For a historical query, it is difficult to guarantee that Greedy Search starting from every entry point will reach the vicinity of the query. Therefore, we fix the entry point as the centroid, which also helps reduce the length of the search path. (2) **Expand the candidate neighbor set of $ANN_{1,q}$** . First, we perform a Greedy Search to retrieve the $ANN_{1,q}$ to the historical query (line 2). If the search did not reach the vicinity of the query (line 3), we expand the candidate neighbors of $ANN_{1,q}$. First, we collect all points u that satisfy $\delta(u, q) < \delta(ANN_{1,q}, q)$ as the extended candidate neighbor set (line 4). Then, we apply the RNG pruning strategy to this set to ensure that the angle between any two connected edges is greater than 60° (lines 5-9), which disperses the edges in different directions and enhances the navigational performance of the graph. Since these edges provide important paths to reach the query, their EH is set to infinity (line 9). Finally, we add these edges to the graph using the same EH pruning strategy as in NGFix (line 10). For the computation of extended candidate neighbor set (line 4), brute-force searching would be time-consuming. Therefore, we use a greedy search with a larger search list size and check all the points visited during the search process to replace the brute-force search.

Since a single RFix does not guarantee that the search will reach the vicinity of the query, we repeat the RFix until the search can reach the query's vicinity or the edge degree limit for that point is reached. As only a small number of historical queries encounter issues where the search cannot reach the query's vicinity, RFix does not significantly increase the graph construction time or index size. After applying both NGFix and RFix, we have the following theorem.

THEOREM 5. *Let T denote the set of historical queries. Without pruning the edges, our method ensures that when $q \in T$ and $k \leq N_q$, the accuracy of GreedySearch(G, q, k , centroid, K_h) is 100% (Algorithm 1).*

Recall that MRNG guarantees the accuracy of Greedy Search when $q \in X$ (base data), experiments show that the search accuracy remains high when q is close to X [23]. Our algorithm serves a similar

Algorithm 4 : Reachability Fixing

Input : Query q , Graph index $G = (V, E, E_{EX}), N_q$
Output : New Graph Index G_{EX}

```

1  $ep \leftarrow$  centroid of base data
2  $ANN_{1,q} \leftarrow GreedySearch(G, q, 1, ep, N_q)$  // Algorithm 1
3 if  $\delta(ANN_{1,q}, q) > \delta(N_{N_q,q}, q)$  then // can not reach query
4    $S \leftarrow \{u | u \in X, \delta(u, q) < \delta(ANN_{1,q}, q)\}$ 
5   Sort points in  $S$  in ascending order of their distance to  $ANN_{1,q}$ .
6    $Candidates \leftarrow \emptyset$  // candidate neighbor set
7   for  $v \in S$  do // MRNG-based pruning
8     if  $\forall r \in Candidates, \delta(v, r) > \delta(ANN_{1,q}, v)$  then
9        $Candidates \leftarrow Candidates \cup \{(ANN_{1,q}, v, inf)\}$ 
10   $E_{EX} \leftarrow E_{EX} \cup Candidates$ 
11  // Use the same EH pruning strategy as in lines 11-15 of Algorithm 3.
12 return  $G_{EX} = (V, E, E_{EX})$ 

```

purpose as MRNG, but the key difference is that we construct the graph on the query distribution, whereas MRNG constructs the graph on the base data distribution. The reasons for setting an out-degree limit for each point are as follows: (1) to prevent the graph index from becoming too large; (2) an excessive number of edges will benefit historical queries, but it does not necessarily improve the performance of other queries and may increase the computational burden during the search process.

5.5 Index Maintenance

We have introduced how to dynamically update the graph structure based on queries. However, dynamic addition and deletion of base data is also a critical issue in real-world applications [60, 74]. Next, we discuss how the graph index handles insertion and deletion operations on the base data.

5.5.1 Insertion. To support dynamic insertion, we require a base graph structure that allows incremental updates (e.g., HNSW). When a new data point arrives, it is first inserted into the base graph following its insertion algorithm. However, after a large number of insertions, the extra edges linked by NGFix and RFix may have limited impact on newly inserted points, which leads to a degradation in search performance. To address this issue, we propose a partial rebuilding strategy with the following steps: (1) For each node, randomly remove a proportion r (e.g., 20%) of its extra outgoing edges (will not remove the edge of base graph), and reset the EH values of the remaining edges to zero. This is because the previously estimated hardness may no longer reflect the current graph structure accurately. (2) Randomly select $r|T|$ historical queries from the set of historical queries T and apply NGFix and RFix on the updated graph. A smaller r leads to faster reconstruction but may result in lower graph quality. We provide a detailed analysis of the trade-offs in Section 6.7.

5.5.2 Deletion. The most commonly used method for deleting points is lazy deletion [11, 49, 53], where the deleted points are retained for navigation during the search process but are excluded from the final result set. This method is simple and efficient. However, when a large number of points are deleted, the deleted points in the graph can significantly prolong the search paths, resulting in degraded search performance. To address this issue, we need to completely remove the point and its associated edges from the graph. This introduces two main challenges: (1) We need to identify and remove all incoming edges to the deleted point. A trivial solution is to store all incoming edges for

DataSet	$ X $	$ Q $	$ T $	d	Distance	Type
Text-to-Image10M	10M	10K	10M	200	InnerProduct	Text, Image
LAION10M	10M	10K	10M	512	Cosine	Text, Image
WebVid2.5M	2.5M	10K	2.5M	512	Cosine	Text, Video
MainSearch	11.2M	50K	1M	256	InnerProduct	Text, Image
SIFT10M	10M	10K	10M	128	Euclidean	Image
DEEP10M	10M	10K	10M	96	Cosine	Image

Table 1. Statistics of the datasets. $|X|$, $|Q|$ and $|T|$ denote the number of base data, test queries and historical queries, respectively. d denotes the dimensionality of vectors.

each point in memory, but this significantly increases memory consumption. (2) The deletion of the point may damage the structure around it, resulting in reduced connectivity in its neighborhood.

For the first challenge, we adopt a lazy deletion strategy for a small number of deletions. Once the number of deleted points exceeds a certain threshold (e.g., 1% of the base data), we perform a full traversal of the graph index to remove all deleted points along with their incoming edges. For the second challenge, recall that NGFix is designed to fix defective graph structures in the vicinity of queries. Thus, we can simply treat the deleted point as a query and execute NGFix. The connectivity degradation resulting from the point deletion will be mitigated by applying NGFix.

6 EXPERIMENTS

6.1 Experimental Setup

Datasets. We mainly evaluate our method using four modern large-scale cross-modal datasets. In Text-to-Image [9] and LAION [59], the base data are image vectors, while the queries are text vectors. In WebVid [8], the base data are embeddings of videos, and the queries are text vectors. We applied deduplication to the queries, ensuring that the test queries are entirely different from historical queries. We then randomly sampled 10k queries that are different from historical queries for testing. Specifically, the MainSearch dataset is sourced from a large e-commerce platform, where the base data corresponds to products. Each product comprises an image and accompanying text, which we combine and embed into a single vector. Query vectors, on the other hand, are derived from either text or image embeddings. The queries are collected over a continuous time, and we use the last 50k queries that differ from historical queries for testing. For the MainSearch dataset, we primarily evaluate the performance of different graph indexes when historical queries are limited and the performance of our method when constructing the index online. Additionally, we conduct experiments using two single-modal datasets [36, 75]. The statistics of the datasets are listed in Table 1.

In the Text-to-Image dataset, text and image embeddings are generated by a variant of DSSM [33] and SE-ResNeXt-101 [32], respectively, and are mapped into a shared space via contrastive learning. The embeddings in LAION and WebVid are generated using CLIP-ViT-B/32 [56]. In MainSearch, we first use a 3B-parameter multimodal LLM³ to generate 3072-dimensional embeddings, which are then compressed to 256 dimensions using Matryoshka Representation Learning [38]. The embeddings in DEEP and SIFT datasets are generated by GoogLeNet [62] and Scale-Invariant Feature Transform [44], respectively.

Algorithms and Parameter Setting. We use $\mathbb{G}$ -NGFix to denote the graph index after NGFix on base graph $\mathbb{G}$ and we use $\mathbb{G}$ -NGFix* to denote the graph index after NGFix and RFix on $\mathbb{G}$. For cross-modal datasets, since RoarGraph has significantly outperformed existing graph algorithms [11], we mainly compare the performance of HNSW-NGFix* and RoarGraph [11], using HNSW [49] and

³Due to commercial privacy considerations, we are unable to provide further details about the LLM.

NSG [23] as baselines. For single-modal datasets, we compare the performance of HNSW-NGFix*, HNSW, τ -MNG [53], RoarGraph and NSG. We determine the parameters for each graph index through multiple experiments. The parameters are as follows: (1) HNSW-NGFix*: We set $M_{EX} = 48$ to limit the out-degree of each node. For each historical query, we apply NGFix and RFix twice to fix defects in the graph. In the first round of fixing, we set $N_q = 100$, $K_h = 100$ and $MaxS \leq 500$; in the second round, we set $N_q = 10$, $K_h = 10$ and $MaxS \leq 50$. The reason for performing two rounds of fixing is that using $N_q = 100$ mainly improves performance when the search list size L is large (i.e., ≥ 100). However, in scenarios where only a small number of nearest neighbors are required (e.g., retrieving the top 10 NNs), a smaller L is sufficient. To ensure performance in such cases, we apply NGFix and RFix again with $N_q = 10$ ⁴. The number of historical queries used for NGFix and RFix is summarized in Table 1. We use brute-force to calculate the exact k NN of each historical query. For cross-modal datasets, to prevent excessive node out-degrees, we set $M = 16$ and use $efC = 2000$ to obtain high-quality neighbors for each point in base graph. In contrast, for single-modal datasets, where the number of hard queries is relatively small and NGFix adds fewer edges, we set $M = 32$ and $efC = 2000$. Existing research indicates that in uniformly distributed random data, the upper layers of HNSW play a limited role when the dimensionality $d \geq 32$ [51]. In vector retrieval scenarios, d typically exceeds 32, and the upper layers of HNSW also demonstrate limited effectiveness under real-world data distributions [43, 69, 70]. To avoid the additional indexing time and size overhead caused by the upper layers, we use only the bottom layer of HNSW as base graph $\mathbb{G}$. (2) RoarGraph: Similar to $\mathbb{G}$ -NGFix, we set $N_q = 100$ and use the same number of historical queries as $\mathbb{G}$ -NGFIX. Additionally, we set $M = 32$ and $L = 2000$ to obtain high-quality neighbors. (3) HNSW: We set $efC = 2000$ to provide good neighbors for nodes in the graph, and we set $M = 32$ to control the out-degree of each node. (4) NSG: We set $C = 2000$ and $L = 2000$ to provide good quality of neighbors, and we set $R = 64$ to control the average out-degree of the graph. (5) τ -MNG: For the same reasons as NSG, we set $R = 64$, $C = 2000$, $L = 2000$. According to the recommendations in the paper, we set τ to 10 for SIFT10M and τ to 0.01 for DEEP10M.

Evaluation metrics. Following previous works [11, 23, 34, 49, 53], we use average recall@ k and rderr@ k (defined in Section 2) to measure the query accuracy. We use query-per-second (QPS) and Number of Distance Calculations (NDC) to measure efficiency. We compare the QPS at the same recall@ k and the NDC at the same rderr@ k , where a higher QPS or lower NDC indicates a better index. When plotting the recall@ k -QPS curves, we initially set search list size L (a.k.a., ef_search in HNSW) to k , and then incremented it by 10 at each step. For each L , we recorded the corresponding QPS and recall@ k . We select representative points to plot the curves. The rderr@ k -NDC curves were generated in the same manner. We mainly focus on the cases where $k = 10$ and $k = 100$.

We implemented NGFix* in C++ and compiled our code using g++ 10.1.0. Experiments were conducted on an Intel(R) Xeon(R) Gold 5220 CPU@2.20GHz with 256GB DDR4 memory (@2666MT/s) under CentOS7.9. We used the same distance computation functions for different graph indexes. All queries were performed with a single thread. Additionally, we utilize 32 threads when evaluating the index construction time of different indexes.

6.2 Search performance

Cross-Modal Datasets. Figure 8 presents the QPS–recall@100 and NDC–rderr@100 curves for various graph indexes across four cross-modal datasets. HNSW-NGFix* outperforms the other algorithms: at recall@100=0.95, its QPS is 1.3–1.6 times that of RoarGraph and 1.7–3.66 times that of HNSW. At recall@100=0.99, HNSW-NGFix*'s advantage becomes even more pronounced,

⁴For RFix, setting $N_q = 10$ also ensures that the search can reach the vicinity of the query when $N_q = 100$. Therefore, RFix only needs to be performed once with $N_q = 10$.

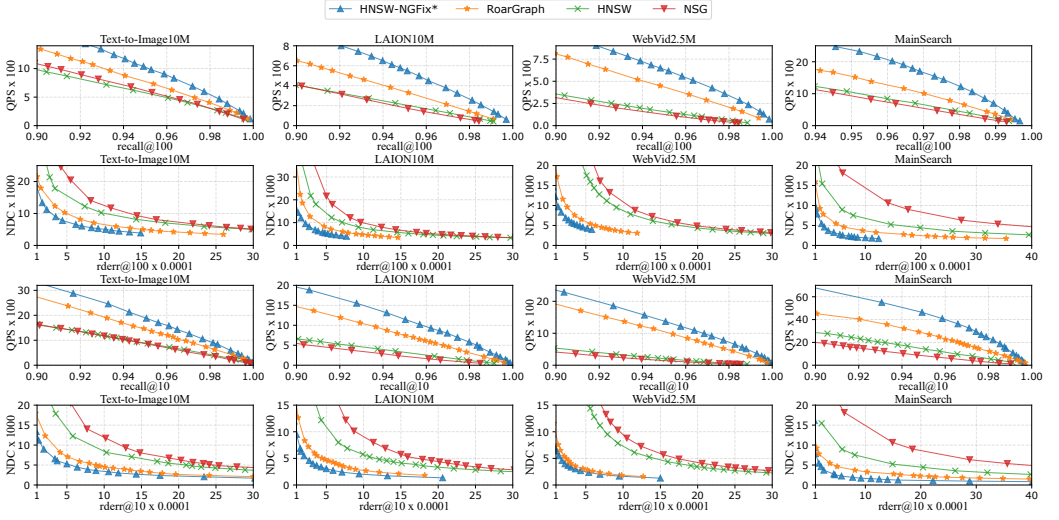


Fig. 8. Search performance on cross-modal datasets.

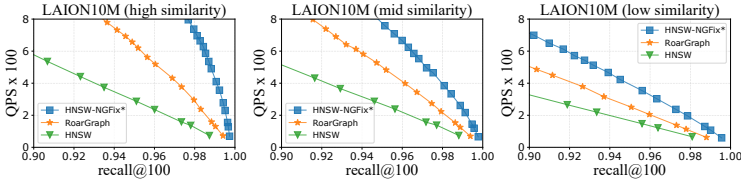


Fig. 9. Evaluation of queries with different similarities.

with a QPS 1.31–2.25 times higher than RoarGraph and 1.78–6.88 times higher than HNSW. In terms of NDC and $\text{rderr}@100$, HNSW-NGFix* also demonstrates significant superiority, especially when aiming for low error rates. When $\text{rderr}@100$ is less than 0.0001, HNSW-NGFix* requires only 48.5%–78% of the distance calculations needed by RoarGraph. For the case of $k = 10$, HNSW-NGFix* also achieves better search performance compared to other graph indexes. Besides the performance improvement, we have the following additional insights. (1) On the MainSearch dataset, we can still achieve significantly higher performance than HNSW using only a small number of historical queries. This is because queries in real-world production environments often exhibit similar characteristics, making it more likely for the test queries to fall within regions of the graph where defects have already been fixed. (2) Compared to RoarGraph, HNSW-NGFix* shows more significant performance advantages when achieving high recall (e.g., $\text{recall}@100 > 0.98$) or low rderr (e.g., $\text{rderr}@100 < 0.0001$). The reason lies in our use of EH to guide graph construction, which takes advantage of the hard queries with poor neighboring structures. The edges linked by these queries make the final graph index more effective in achieving high recall.

OOD Queries with Different Similarities. We evaluate our method on test queries with varying similarity to historical queries. We measure query similarity using the distance d between test queries and their nearest historical queries. In LAION10M (distance is defined as $1 - \cos$), 18.1% of the test queries have $d \leq 0.05$ (i.e., high similarity), and 26.2% satisfy $0.05 < d \leq 0.1$ (i.e., moderate similarity). The remaining queries exhibit low similarity. The thresholds depend on the characteristics of each dataset. For LAION10M, we choose 0.05 and 0.1 to better illustrate

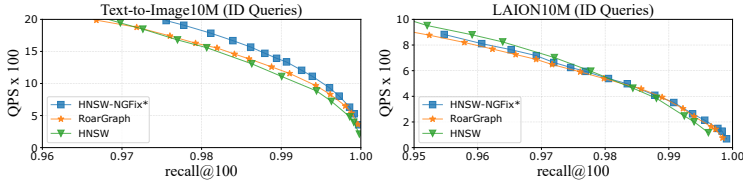


Fig. 10. Evaluation of ID Queries in cross-modal datasets.

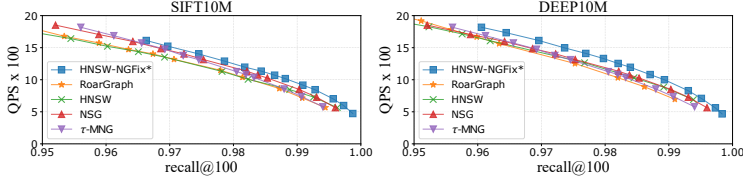


Fig. 11. Search performance on single-modal datasets.

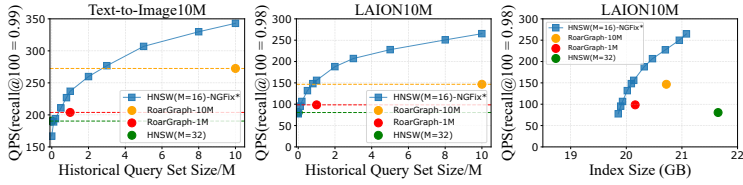


Fig. 12. Evaluation of historical query set size.

the relationship between index performance and query similarity. Figure 9 demonstrates that our method outperforms others under varying similarity levels.

ID Queries in Cross-Modal Datasets. In cross-modal retrieval, both OOD queries (e.g., text-to-image) and ID queries (e.g., image-to-image) may occur. As shown in Figure 10, the index refined by NGFix* with OOD queries also achieves strong performance on ID queries. This indicates that fixing with OOD queries does not affect the performance of ID queries.

Single-Modal Datasets. Figure 11 shows the QPS-recall@100 curves in single-modal datasets. Our approach achieves modest QPS improvements of approximately 10%. This limited improvement is due to NGFix*'s emphasis on hard queries, which are rare in single-modal datasets, leading to fewer effective edge connections. Another reason is that HNSW and NGFix* construct the graph based on the base data distribution and the query distribution, respectively. However, for single-modal datasets, these two distributions are consistent. Since HNSW already fits the base data distribution well and only a few hard queries that are relatively far from the base data benefit from our method, the improvement brought by NGFix* is limited in this case. On the other hand, RoarGraph's graph construction strategy fails to leverage the information provided by these hard queries. In some cases, the edges even increase the computational burden during the search, causing its performance to be even worse than HNSW.

Since the single-modal datasets contain fewer hard queries and we can easily achieve high accuracy, our subsequent experiments primarily focus on evaluating the performance of our method on cross-modal datasets.

6.3 Effect of Historical Query Set Size.

In this experiment, we evaluate the performance of our method with varying numbers of historical queries. Since RoarGraph needs to entirely rebuild the graph when the number of historical queries

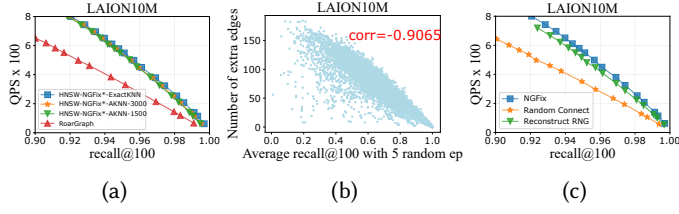


Fig. 13. (a) Evaluation of different preprocessing methods. (b) Evaluation of the Escape Hardness. (c) Evaluation of different defects fixing methods.

differs from the previous configuration, building RoarGraph with different quantities of historical queries demands substantial time. So we only selected RoarGraph with 1M and 10M historical queries for comparison and we denote RoarGraph with p historical queries as RoarGraph- p . Figure 12 illustrates the QPS achieved by HNSW-NGFix* at specific recall@100 across different numbers of historical queries. HNSW-NGFix* requires only 8%-30% of the historical queries compared to RoarGraph-10M when achieving the same performance on different datasets. Additionally, HNSW (M=16)-NGFix*, using merely 1% of the historical queries relative to the base data size, achieves the same performance as HNSW (M=32). These results demonstrate that our method can achieve superior performance with fewer historical queries and can attain high performance in a short time when constructing the graph online. The rightmost subfigure in Figure 12 shows the trade-off between index size and QPS. We vary the number of historical queries used to control the index size of HNSW-NGFix*. The results indicate that HNSW-NGFix* achieves higher QPS compared to both RoarGraph and HNSW under the same index size.

6.4 Ablation Study

Evaluating the impact of different preprocessing methods. We compare the graph quality achieved by using exact k NN against that of approximate k NN (described in section 5.1). HNSW-NGFix-ExactKNN represents the graph constructed using exact k NN obtained through brute-force methods for historical queries. Conversely, HNSW-NGFix-AKNN- L denotes the graph built using approximate k NN derived from a Greedy Search with a search list size of L . As illustrated in Figure 13a, our experimental results show that the QPS-recall@100 curves for approximate k NN are almost identical to those of exact k NN, with only a slight decrease.

Evaluating the effectiveness of EH in differentiating between easy and hard queries. We evaluate whether Escape Hardness could guide us in linking different numbers of edges according to the hardness of the queries. Figure 13b reveals a high correlation between query accuracy and the number of additional edges connected by NGFix. This indicates that our algorithm concentrates on hard queries by adding more edges for them while linking fewer edges for easy queries. Consequently, our approach not only optimizes the graph index size but also reduces the overhead of distance calculations during searches.

Evaluating different defects fixing methods. Figure 13c illustrates the performance of graphs generated by different defects fixing methods (introduced in Figure 7). Among these, random connecting exhibits the poorest performance because randomly connected edges fail to provide effective neighbors for each node. Meanwhile, NGFix achieves better performance than the method of reconstructing RNG. Moreover, the average out-degree of the graph derived from Reconstructing RNG is approximately 1.37 times that of NGFix, indicating that NGFix reduces the index size while maintaining a high-quality graph structure.

Evaluating the impact of different edge-pruning strategies. Figure 14 illustrates the performance of different edge-pruning strategies employed in NGFix (Algorithm 3 lines 11-15).

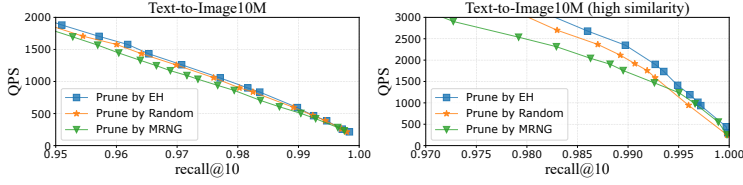


Fig. 14. Evaluation of different edge-pruning strategies.

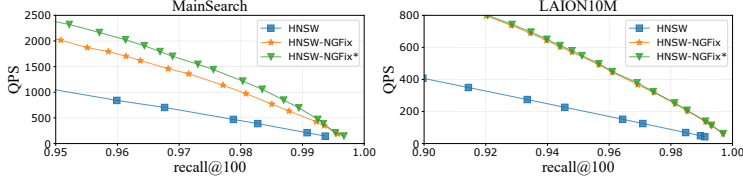


Fig. 15. Evaluation of Reachability Fixing

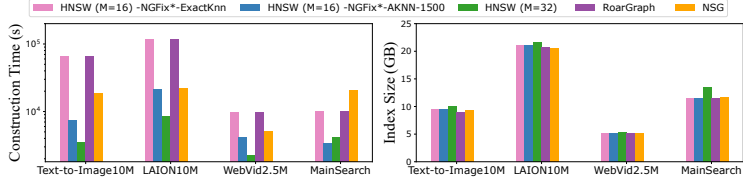


Fig. 16. Construction time and index size

Pruning based on EH achieves better performance than random pruning, whereas MRNG pruning exhibits the poorest performance. This is because MRNG pruning is designed for queries that overlap with the base data [23] and is more likely to prune long edges (i.e., edge (u, v) with large $\delta(u, v)$). However, since the k NN of hard queries are often scattered across different regions [11], these long edges are often very useful for hard queries, and pruning them adversely affects the connectivity of the graph around hard queries, leading to significant losses in accuracy and performance.

Evaluating the impact of the RFix. Figure 15 shows the performance of NGFix and NGFix*. In the MainSearch dataset, due to the relatively large proportion of searches that cannot reach the query vicinity (Figure 2b), NGFix* improves performance by 18% over NGFix when recall@100=0.95. In LAION10M, since the search for almost all queries can reach the query vicinity before RFix, NGFix* only shows a slight improvement over NGFix.

6.5 Construction Time and Index Size

Figure 16 reports the index construction time and the index size across different datasets. The index size reflects the memory consumption during the search. When using the exact k NN for graph construction, the indexing time of HNSW-NGFix* is almost the same as that of RoarGraph. However, when a large number of historical queries are involved, the construction time becomes significantly higher than that of other methods. When using approximate k NN, HNSW-NGFix* accelerates index construction by 2.35-9.02 times compared to RoarGraph (RoarGraph lacks a complete graph index when using historical queries, which makes it unable to leverage approximate k NN for graph construction). Meanwhile, HNSW-NGFix* requires 82.1% to 210.2% of the indexing time taken by HNSW, and 16.5% to 95.5% compared to NSG. On the MainSearch dataset, HNSW achieves its best performance with $M = 32$, while HNSW-NGFix* only needs $M = 16$, resulting in a shorter index construction time than HNSW. In terms of index size, HNSW-NGFix* is smaller than HNSW since

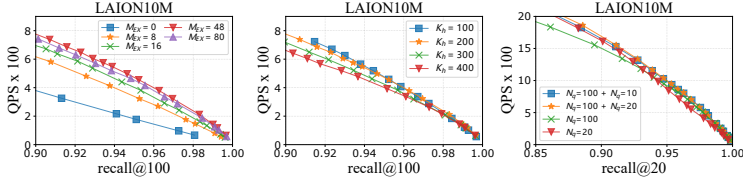


Fig. 17. Evaluation of different parameters.

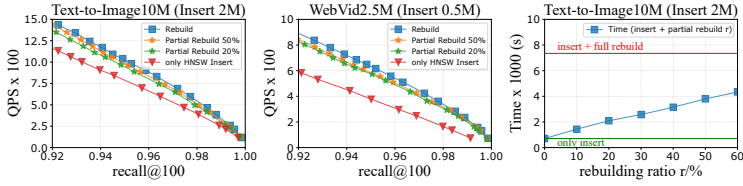


Fig. 18. Evaluation of insertion method

it only utilizes the bottom layer of HNSW. However, as we store the EH of each edge (with an additional 16 bits per extra edge) for edge pruning, the index size of HNSW-NGFix* is slightly larger than that of RoarGraph and NSG.

6.6 Effect of Different Parameters

Figure 17 illustrates the performance of the graph index under different parameter settings. For the parameter M_{EX} , we fix $K_h = 200$ and perform a single NGFix with $N_q = 100$ to evaluate the impact of M_{EX} . Since a smaller M_{EX} leads to the pruning of useful edges, while a larger M_{EX} increases the search overhead, we recommend setting M_{EX} in the range of 32–64. For datasets with higher average query hardness (as measured by EH), a larger M_{EX} may be appropriate. Nevertheless, setting $M_{EX} = 48$ generally yields sufficiently good performance. For the parameter K_h , we fix $M_{EX} = 48$ and also perform a single NGFix with $N_q = 100$ to evaluate the impact of K_h . Experimental results show that a smaller K_h tends to produce a more effective index, but at the cost of increased index size. For example, when $K_h = 100$ and $K_h = 400$, the average out-degrees are 41.26 and 30.85, respectively. Therefore, K_h can be selected based on the trade-off between space and performance requirements. For the parameter N_q , we recommend first setting $N_q = 100$, and then performing NGFix* again with $N_q = k$ based on the number of NNs k to be retrieved. The experimental results in the rightmost subfigure of Figure 17 show that when $k = 20$, performing NGFix* with $N_q = 20$ yields slightly better performance. Nonetheless, in scenarios where k varies frequently, it is not necessary to perform NGFix* for all values of k . Performing NGFix* twice with $N_q = 10$ and $N_q = 100$ is sufficient to produce a high-quality index. We include more details about N_q in Appendix C. For the parameter $MaxS$, since very few edges have an EH value exceeding $5N_q$, setting $MaxS$ to $5N_q$ is adequate. If N_q is set to a large value (e.g., ≥ 300), it is recommended to set $MaxS$ to $1.2N_q - 2N_q$ to reduce the index construction overhead, as lowering $MaxS$ will not significantly affect the index quality. For the parameter settings of the base graph, we can follow the selection strategy of the corresponding graph index. Meanwhile, slightly reducing the average out-degree is recommended to avoid an overly large final graph (e.g., reducing HNSW’s M by 16).

6.7 Evaluation of Index Maintenance Methods

In this experiment, to ensure a fair comparison with rebuilding, we perform both insertion and deletion operations using 32 threads.

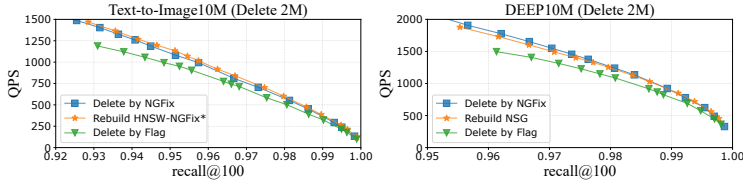


Fig. 19. Evaluation of deletion method

Insertion. Figure 18 shows the performance of the graph index after inserting 20% of the data points using different insertion algorithms. “Partial Rebuild r ” in the figure denotes a partial rebuilding with proportion r , as described in Section 5.5.1. We perform a partial rebuilding after inserting 20% of the data. Compared with directly using HNSW’s insertion algorithm, partial rebuilding significantly improves the quality of the resulting graph index. A higher proportion leads to better index quality. The rightmost subfigure in Figure 18 shows the relationship between time (insertion + rebuilding) and the parameter r . When $r = 20\%$, the total time is only 28.5% of that for full rebuilding. In practice, we can choose an appropriate proportion based on time and quality requirements. If new points are inserted during partial rebuilding, the insertion and rebuilding processes remain independent. This is because partial rebuilding only modifies the extra edges, while inserting new points only modifies the base graph. So the total insertion time is not affected.

Deletion. When performing deletions with NGFix, we adopt greedy search with a search list size of 800 to obtain the approximate ground truth, and set parameters $N_q = 100$, $K_h = 100$, $MaxS \leq 500$. Figure 19 presents the results of the deletion experiment. The left panel shows the performance of different methods after removing 20% of the base data on the Text-to-Image dataset. It can be observed that the performance of the index degrades significantly under the lazy deletion strategy. In contrast, the index quality achieved by applying NGFix after deletion is nearly identical to that of full reconstruction, with only a slight drop. The time cost of deletion using NGFix is only 6.8% of that of a full rebuilding. To demonstrate the robustness of our deletion algorithm, the right panel shows the performance after deletion using NGFix on an NSG index (without NGFix* based on historical queries). It can be observed that the performance after deletion even surpasses that of full reconstruction, as NGFix is capable of linking edges of higher quality than those in the original NSG. This demonstrates that our proposed deletion method also performs well on other graph indexes.

7 DISCUSSION AND FUTURE WORK

Mitigate the dependence on historical queries. In certain scenarios (e.g., cold-start), the lack of sufficient historical queries may degrade index performance. We propose two strategies to mitigate this issue: (1) Since multimodal embedding models typically require a large amount of multimodal data for training, we can use the training data as historical queries. (2) In scenarios involving workload drift, only a few representative queries may be available. To address this limitation, we propose a data augmentation method to synthetically generate a larger set of queries. We define HNSW-NGFix*- $p\%-q\%$ to use some real historical queries equal to $p\%$ of the base data size, along with generated queries equal to $q\%$ of the base data size. For each real historical query, we generate q/p synthetic queries. In each generation, Gaussian noise with zero mean and variance $\sqrt{c/d}$ is added independently to every dimension of historical queries (d represents the dimensionality). In experiments on WebVid and MainSearch, we selected $c = 0.3$ as it achieved the best performance among $c = 0.1, 0.2, 0.3, 0.4$. Experimental results in Figure 20 demonstrate the effectiveness of this method.

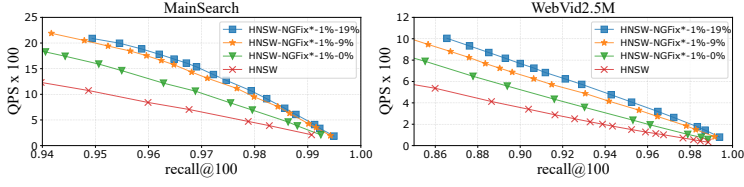


Fig. 20. Evaluation of generated historical queries.

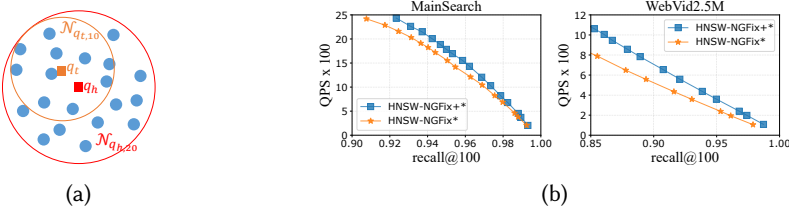


Fig. 21. (a) Example of NGFix+. (b) Evaluation of NGFix+.

Selection strategy for historical queries. When the query workload remains stable, it is unnecessary to design a historical query selection strategy, as NGFix* can adaptively adjust the graph index based on the query hardness. For queries with low EH, NGFix* adds very few edges or skips them altogether. In contrast, for high-EH queries, it adds more edges to effectively fix defects in the graph. So it is sufficient to use all available historical queries. In scenarios with dynamic workloads, the limited out-degree of each node may prevent new edges from being added for incoming queries, as edges added for old historical queries may occupy the out-degree budget. To prioritize fixing the graph structure for new queries, we periodically delete a subset of existing extra edges added by NGFix* to make room for newly added edges (e.g., by randomly selecting 20% extra edges). Then, we prioritize using the newest queries based on their timestamps to fix the graph. This allows the index to better adapt to the new workload. Exploring better strategies to adapt to workload changes could be considered a direction for future work.

Extending the theoretical guarantee. Let $N_{q,k}$ denote the set of the top- k NN of query q . Let q_h denote historical query, and q_t denote the new test query. One direction for extending the theoretical guarantee is to ensure query accuracy when $\delta(q_h, q_t) < \epsilon$. We let U_k denote the union of the top- k NN sets of all queries q_t that satisfy $\delta(q_h, q_t) < \epsilon$. For each q_h and a fixed ϵ , in order to guarantee the accuracy of all queries satisfying $\delta(q_h, q_t) < \epsilon$, we need to consider all points in the set U_k . Since $|U_k|$ can be very large (in the worst case, it could be the size of the base data), ensuring query accuracy becomes very difficult. Therefore, we try to extend the theoretical guarantee from another perspective. Assuming that the search can reach the vicinity of the query, one direction for theoretical extension is as follows: **when $N_{q_t,k} \subseteq N_{q_h,K}$, we aim to guarantee that the recall@ k for q_t is 100%, where search list size $L \geq k$, $K = ck$ and c is a constant (e.g., 2).** Figure 21a provides an example of this theory. Our method (NGFix) is already able to ensure accuracy where $c = 1$ (i.e., $N_{q_t,k} = N_{q_h,k}$). In the WebVid dataset, experimental results show that when $N_{q_t,10} = N_{q_h,10}$, $\delta(q_t, q_h)$ is approximately less than 0.03 (Euclidean distance). Moreover, when $N_{q_t,10} \subseteq N_{q_h,20}$, $\delta(q_t, q_h)$ is approximately less than 0.114. Details of this experiment are provided in Appendix D.

To ensure the property when $c > 1$, a trivial approach is to enumerate all $N_{q_t,k}$ and apply NGFix for each one. We conducted experiments on cross-modal datasets with parameters set to $k = 100$ and $K = 200$ (i.e., $c = 2$). To avoid excessive graph construction overhead, we use a small number of historical queries (2.5×10^4 in WebVid and 10^5 in MainSearch) and randomly enumerate $100N_{q_t,100}$

for each historical query. We denote the new method as NGFix+, and the experimental results in Figure 21b show that NGFix+ outperforms NGFix. However, NGFix+ takes 19.2 times longer to run than NGFix. Therefore, one direction for future work is to develop more efficient algorithms to ensure the above property.

Hash-Table Method. When test queries exactly overlap with historical queries, we can use a hash-table method. The main steps are as follows: (1) Apply a hash function (e.g., MD5 [58]) to each query to obtain a compact key, and store the key along with the corresponding ground truth for historical queries in a hash table. (2) For a test query, if it can be retrieved from the hash table, directly return the ground truth; otherwise, use ANNS. The latency of the hash-table method is about 9.3% of that of graph-based ANNS in MainSearch dataset (using HNSW-NGFix* with search list size $L = 100$). However, the hash-table method cannot be generalized to unseen queries. Moreover, this method requires substantial extra space to store the ground truth and may suffer from errors due to hash collisions. In scenarios with high query repetition, using both hash table and HNSW-NGFix* can achieve higher QPS.

Query Similarities. In graph-based ANNS, a fixed search parameter L is typically used for all queries when aiming for a predetermined target recall [11, 23, 28, 49]. However, the experimental results in Figure 9 show that the value of L required (i.e., the latency required) to achieve a specific recall varies significantly across queries with different similarity levels. This observation suggests the potential for the following adaptive strategy: We first compute the similarity between the new query and historical queries, and then adjust the search parameter L based on the similarity. Hence, one direction for future work is to develop methods for efficient query similarity calculation and fine-grained adjustment of L .

8 CONCLUSION

In this paper, we introduce NGFix and RFix, which are designed to improve graph index quality by exploiting the distribution of queries, with some theoretical foundations. Following prior work [65, 70], we divide the search process into two stages: (1) In the stage of approaching the query from the entry point, we analyze the limitations of existing approximation strategies for graph construction and introduce RFix to enhance the navigability of certain nodes; (2) In the stage of searching around the query, we first propose Escape Hardness to evaluate the quality of the graph index in the vicinity of the query. We then propose NGFix to dynamically fix defective regions of the graph based on Escape Hardness. Our method outperforms state-of-the-art methods on real-world datasets and significantly improves the accuracy of current hard queries.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and suggestions. This research is supported by the National Natural Science Foundation of China (grant numbers, 62272252, 62272253), and the Fundamental Research Funds for Central Universities.

References

- [1] 2024. *Minimum spanning tree*. https://en.wikipedia.org/wiki/Minimum_spanning_tree Accessed on January 7, 2025.
- [2] Salam Albatarni, Sohaila Eltanbouly, and Tamer Elsayed. 2024. Graded Relevance Scoring of Written Essays with Dense Retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Washington DC, USA) (SIGIR '24). Association for Computing Machinery, New York, NY, USA, 1329–1338. doi:10.1145/3626772.3657744
- [3] 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 (Dec. 2015), 288–299. doi:10.14778/2856318.2856324

- [4] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. HD-index: pushing the scalability-accuracy boundary for approximate kNN search in high-dimensional spaces. *Proc. VLDB Endow.* 11, 8 (April 2018), 906–919. doi:10.14778/3204028.3204034
- [5] Franz Aurenhammer. 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.* 23, 3 (Sept. 1991), 345–405. doi:10.1145/116873.116880
- [6] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. ELPIS: Graph-Based Similarity Search for Scalable Data Science. *Proc. VLDB Endow.* 16, 6 (Feb. 2023), 1548–1559. doi:10.14778/3583140.3583166
- [7] 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 (Feb. 2025), 31 pages. doi:10.1145/3709693
- [8] Max Bain, Arsha Nagrani, Gül Varol, and Andrew Zisserman. 2021. Frozen in Time: A Joint Video and Image Encoder for End-to-End Retrieval. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 1708–1718. doi:10.1109/ICCV48922.2021.00175
- [9] Artem Babenko Dmitry Baranchuk. 2021. *Text-to-Image dataset for billion-scale similarity search*. <https://research.yandex.com/datasets/text-to-image-dataset-for-billion-scale-similarity-search> Accessed on December 19, 2024.
- [10] Dmitry Baranchuk, Dmitry Persiyonov, Anton Sinitin, and Artem Babenko. 2019. Learning to Route in Similarity Graphs. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 475–484. <https://proceedings.mlr.press/v97/baranchuk19a.html>
- [11] Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X. Sean Wang. 2024. RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment* 17, 11 (July 2024), 2735–2749. doi:10.14778/3681954.3681959
- [12] 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* (Austin, TX, USA) (WWW '23). Association for Computing Machinery, New York, NY, USA, 3225–3235. doi:10.1145/3543507.3583318
- [13] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) (RecSys '16). Association for Computing Machinery, New York, NY, USA, 191–198. doi:10.1145/2959100.2959190
- [14] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (Brooklyn, New York, USA) (SCG '04). Association for Computing Machinery, New York, NY, USA, 253–262. doi:10.1145/997817.997857
- [15] B. Delaunay. 1933. Neue Darstellung der geometrischen Kristallographie. *Zeitschrift für Kristallographie - Crystalline Materials* 84, 1-6 (1933), 109–149. doi:10.1524/zkri.1933.84.1.109
- [16] Liwei Deng, Penghao Chen, Ximu Zeng, Tianfu Wang, Yan Zhao, and Kai Zheng. 2025. Efficient Data-Aware Distance Comparison Operations for High-Dimensional Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 18, 3 (April 2025), 812–821. doi:10.14778/3712221.3712244
- [17] Matthijs Douze and Hervé Jégou. 2014. The Yael Library. In *Proceedings of the 22nd ACM International Conference on Multimedia* (Orlando, Florida, USA) (MM '14). Association for Computing Machinery, New York, NY, USA, 687–690. doi:10.1145/2647868.2654892
- [18] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) (KDD '24). Association for Computing Machinery, New York, NY, USA, 6491–6501. doi:10.1145/3637528.3671470
- [19] Chao Feng, Defu Lian, Xiting Wang, Zheng Liu, Xing Xie, and Enhong Chen. 2023. Reinforcement Routing on Proximity Graph for Efficient Recommendation. *ACM Trans. Inf. Syst.* 41, 1, Article 8 (Jan. 2023), 27 pages. doi:10.1145/3512767
- [20] Robert W. Floyd. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5, 6 (June 1962), 345. doi:10.1145/367766.368168
- [21] Cong Fu and Deng Cai. 2016. EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph. arXiv:1609.07228 [cs.CV] <https://arxiv.org/abs/1609.07228>
- [22] Cong Fu, Changxu Wang, and Deng Cai. 2022. High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2022), 4139–4150. doi:10.1109/TPAMI.2021.3067706
- [23] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.* 12, 5 (Jan. 2019), 461–474. doi:10.14778/3303753.3303754
- [24] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2025. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 3, Article 202 (June 2025), 26 pages. doi:10.1145/3725413

- [25] 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 (June 2023), 27 pages. doi:10.1145/3589282
- [26] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 3, Article 167 (May 2024), 27 pages. doi:10.1145/3654970
- [27] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 4 (2014), 744–755. doi:10.1109/TPAMI.2013.240
- [28] Yutong Gou, Jianyang Gao, Yuexuan Xu, and Cheng Long. 2025. SymphonyQG: Towards Symphonious Integration of Quantization and Graph for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 1, Article 80 (Feb. 2025), 26 pages. doi:10.1145/3709730
- [29] 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 *Proceedings of the 37th International Conference on Machine Learning (ICML '20)*. JMLR.org, Article 364, 10 pages.
- [30] Ben Harwood and Tom Drummond. 2016. FANNG: Fast Approximate Nearest Neighbour Graphs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5713–5722. doi:10.1109/CVPR.2016.616
- [31] Nico Hezel, Kai Uwe Barthel, Konstantin Schall, and Klaus Jung. 2023. Fast Approximate Nearest Neighbor Search with a Dynamic Exploration Graph using Continuous Refinement. arXiv:2307.10479 [cs.LR] <https://arxiv.org/abs/2307.10479>
- [32] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-Excitation Networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7132–7141. doi:10.1109/CVPR.2018.00745
- [33] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management (San Francisco, California, USA) (CIKM '13)*. Association for Computing Machinery, New York, NY, USA, 2333–2338. doi:10.1145/2505515.2505665
- [34] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. 2022. OOD-DiskANN: Efficient and Scalable Graph ANNS for Out-of-Distribution Queries. arXiv:2211.12850 [cs.LG] <https://arxiv.org/abs/2211.12850>
- [35] J.W. Jaromczyk and G.T. Toussaint. 1992. Relative neighborhood graphs and their relatives. *Proc. IEEE* 80, 9 (1992), 1502–1517. doi:10.1109/5.163414
- [36] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. doi:10.1109/TPAMI.2010.57
- [37] L. V. Kantorovich. 1960. Mathematical Methods of Organizing and Planning Production. *Management Science* 6, 4 (1960), 366–422.
- [38] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. 2022. Matryoshka Representation Learning. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 30233–30249. https://proceedings.neurips.cc/paper_files/paper/2022/file/c32319f4868da7613d78af9993100e42-Paper-Conference.pdf
- [39] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 9459–9474. https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf
- [40] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. 2020. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (Portland, OR, USA) (SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2539–2554. doi:10.1145/3318464.3380600
- [41] Haitao Li, Qingyao Ai, Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Zheng Liu, and Zhao Cao. 2023. Constructing Tree-based Index for Efficient and Effective Dense Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (Taipei, Taiwan) (SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 131–140. doi:10.1145/3539618.3591651
- [42] Victor Weixin Liang, Yuhui Zhang, Yongchan Kwon, Serena Yeung, and James Y Zou. 2022. Mind the Gap: Understanding the Modality Gap in Multi-modal Contrastive Representation Learning. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 17612–17625.
- [43] Peng-Cheng Lin and Wan-Lei Zhao. 2019. Graph based Nearest Neighbor Search: Promises and Failures. arXiv:1904.02077 [cs.LR] <https://arxiv.org/abs/1904.02077>

- [44] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision* 60, 2 (2004), 91–110.
- [45] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proc. VLDB Endow.* 15, 2 (Oct. 2021), 246–258. doi:10.14778/3489496.3489506
- [46] Kejing Lu, Chuan Xiao, and Yoshiharu Ishikawa. 2024. Probabilistic Routing for Graph-Based Approximate Nearest Neighbor Search. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 33177–33195. <https://proceedings.mlr.press/v235/lu24l.html>
- [47] Prasanta Chandra Mahalanobis. 1936. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India* 2 (1936), 49–55.
- [48] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68. doi:10.1016/j.is.2013.10.006
- [49] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836. doi:10.1109/TPAMI.2018.2889473
- [50] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin’ichi Satoh. 2018. [Invited Paper] A Survey of Product Quantization. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 2–10. doi:10.3169/mta.6.2
- [51] Blaise Munyampirwa, Vihan Lakshman, and Benjamin Coleman. 2025. Down with the Hierarchy: The ‘H’ in HNSW Stands for ‘Hubs’. arXiv:2412.01940 [cs.LG] <https://arxiv.org/abs/2412.01940>
- [52] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (Virtual Event, CA, USA) (KDD ’20)*. Association for Computing Machinery, New York, NY, USA, 2311–2320. doi:10.1145/3394486.3403280
- [53] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. *Proc. ACM Manag. Data* 1, 1, Article 54 (May 2023), 27 pages. doi:10.1145/3588908
- [54] Zhen Peng, Minjia Zhang, Kai Li, Ruoming Jin, and Bin Ren. 2023. iQAN: Fast and Accurate Vector Search with Efficient Intra-Query Parallelism on Multi-Core Architectures. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (Montreal, QC, Canada) (PPoPP ’23)*. Association for Computing Machinery, New York, NY, USA, 313–328. doi:10.1145/3572848.3577527
- [55] 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)*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). Association for Computational Linguistics, Doha, Qatar, 1532–1543. doi:10.3115/v1/D14-1162
- [56] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 8748–8763. <https://proceedings.mlr.press/v139/radford21a.html>
- [57] Jie Ren, Minjia Zhang, and Dong Li. 2020. HM-ANN: efficient billion-point nearest neighbor search on heterogeneous memory. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (Vancouver, BC, Canada) (NIPS ’20)*. Curran Associates Inc., Red Hook, NY, USA, Article 895, 13 pages.
- [58] Ronald L. Rivest. 1992. The MD5 Message-Digest Algorithm. RFC 1321. doi:10.17487/RFC1321
- [59] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. 2021. LAION-400M: Open Dataset of CLIP-Filtered 400 Million Image-Text Pairs. arXiv:2111.02114 [cs.CV] <https://arxiv.org/abs/2111.02114>
- [60] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search. arXiv:2105.09613 [cs.IR] <https://arxiv.org/abs/2105.09613>
- [61] Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2019. *DiskANN: fast accurate billion-point nearest neighbor search on a single node*. Curran Associates Inc., Red Hook, NY, USA.
- [62] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–9. doi:10.1109/CVPR.2015.7298594
- [63] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2024. DB-LSH 2.0: Locality-Sensitive Hashing With Query-Based Dynamic Bucketing. *IEEE Transactions on Knowledge and Data Engineering* 36, 3 (2024), 1000–1015. doi:10.1109/TKDE.2023.

3295831

- [64] Warren S. Torgerson. 1952. Multidimensional scaling: I. Theory and method. *Psychometrika* 17, 4 (01 Dec 1952), 401–419. doi:10.1007/BF02288916
- [65] Hongya Wang, Zhizheng Wang, Wei Wang, Yingyuan Xiao, Zeng Zhao, and Kaixiang Yang. 2020. A Note on Graph-Based Nearest Neighbor Search. arXiv:2012.11083 [cs.LG] <https://arxiv.org/abs/2012.11083>
- [66] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. 2014. Trinary-Projection Trees for Approximate Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 2 (2014), 388–403. doi:10.1109/TPAMI.2013.125
- [67] 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. *Proc. ACM Manag. Data* 2, 1, Article 14 (March 2024), 27 pages. doi:10.1145/3639269
- [68] 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 (July 2021), 1964–1978. doi:10.14778/3476249.3476255
- [69] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph- and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions. *IEEE Data Eng. Bull.* 46 (2023), 3–21.
- [70] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Steiner-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. *Proc. VLDB Endow.* 17, 13 (Sept. 2024), 4668–4682. doi:10.14778/3704965.3704974
- [71] 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 (Feb. 2025), 29 pages. doi:10.1145/3709729
- [72] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 9 (Aug. 2024), 2241–2254. doi:10.14778/3665844.3665854
- [73] 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 (Feb. 2025), 28 pages. doi:10.1145/3709743
- [74] Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, Peng Cheng, and Mao Yang. 2023. SPFresh: Incremental In-Place Update for Billion-Scale Vector Search. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) (SOSP '23). Association for Computing Machinery, New York, NY, USA, 545–561. doi:10.1145/3600006.3613166
- [75] Artem Babenko Yandex and Victor Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2055–2063. doi:10.1109/CVPR.2016.226
- [76] Michihiro Yasunaga, Armen Aghajanyan, Weijia Shi, Rich James, Jure Leskovec, Percy Liang, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2023. Retrieval-augmented multimodal language modeling. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) (ICML '23). JMLR.org, Article 1659, 15 pages.
- [77] Minjia Zhang, Wenhan Wang, and Yuxiong He. 2022. GraSP: Optimizing Graph-based Nearest Neighbor Search with Subgraph Sampling and Pruning. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining* (Virtual Event, AZ, USA) (WSDM '22). Association for Computing Machinery, New York, NY, USA, 1395–1405. doi:10.1145/3488560.3498425
- [78] 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. *Proc. VLDB Endow.* 16, 8 (April 2023), 1979–1991. doi:10.14778/3594512.3594527
- [79] Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S. Jensen. 2020. PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search. *Proc. VLDB Endow.* 13, 5 (Jan. 2020), 643–655. doi:10.14778/3377369.3377374

Received April 2025; revised July 2025; accepted August 2025