

Prior-Free Indexing for Cross-Modal Vector Search

Jiadong Xie

The Chinese University of Hong Kong
jdxie@se.cuhk.edu.hk

Jeffrey Xu Yu

The Hong Kong University of Science
and Technology (Guangzhou)
jeffreyyu@hkust-gz.edu.cn

Yingfan Liu

Xidian University
liuyingfan@xidian.edu.cn

Abstract

Approximate nearest neighbor (ANN) search has been extensively studied to find k approximate nearest neighbors for a query vector q in a high-dimensional dataset, where a data item is represented as a vector. Among them, proximity graphs (PG) have gained popularity as state-of-the-art solutions to ANN search. However, PGs suffer a significant performance drop on cross-modal vector search queries, termed out-of-distribution (OOD) queries. Existing approaches involve OOD queries during index construction to efficiently support them. However, issues still persist. First, they require OOD queries at a scale similar to the base data, together with their high-quality k -ANN or even exact k -ANN results, which is impractical due to scarcity and time cost. Second, these approaches focus on OOD queries but neglect in-distribution (ID) queries, resulting in reduced performance on ID queries. To address these issues, in this paper, we propose PFIndex, a prior-free index for jointly supporting ID and OOD queries: PFIndex requires neither a pre-collected OOD query set nor its high-quality or exact k -NN results before deployment, yet it can consistently provide high-quality k -ANN for OOD queries and reduce the cost of subsequent OOD queries by augmenting the index based on processed OOD queries. Extensive experiments on real-world datasets demonstrate that PFIndex outperforms state-of-the-art approaches on OOD queries, while also maintaining the best performance on ID queries, with index construction up to 3.18 times faster. Moreover, PFIndex reduces the total index memory by up to 36.7% for jointly supporting ID and OOD queries.

PVLDB Reference Format:

Jiadong Xie, Jeffrey Xu Yu, and Yingfan Liu. Prior-Free Indexing for Cross-Modal Vector Search. PVLDB, 21(1): XXX-XXX, 2027.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/Xiejiaodong/Prior-Free-Indexing>.

1 Introduction

In recent years, the emergence of large language models [22] and advancements in machine learning [32] have significantly amplified the need for handling high-dimensional data. Machine learning models transform various data types into vector representations, empowering the high-dimensional data, or vectors, to encapsulate

Table 1: Comparisons of Different Approaches on LAION

Algorithms	Index Cost		QPS at $recall@10 = 0.95$	
	Time (sec)	Size (MB)	ID Queries	OOD Queries
HNSW	4,299	20,761	3,691.45	164.08
RobustVamana	12,063	20,667	1,262.01	179.39
RoarGraph	(14,288+)16,501	20,589	2,468.81	497.89
NGFix	(14,288+)11,792	21,976	1,885.60	642.06
PFIndex	(4,299+)5,391	(20,761+)7,072	3,691.45	843.65

essential information for various data types, such as text [32], images [34], and audio [3]. Given a dataset $X = \{x_1, x_2, \dots, x_n\} \subset \mathbb{R}^d$ consisting of d -dimensional vectors, a fundamental problem is the approximate nearest neighbor (ANN) search, which aims to find k -approximate nearest neighbors in X for a given query vector q . To achieve high accuracy with low latency in ANN searches, various approaches have been proposed [9–12, 17, 19, 23, 25, 31, 33, 35, 38, 41, 44, 47, 48]. Among them, proximity graphs (PG) have gained popularity as state-of-the-art solutions to ANN search, demonstrating the best performance on many scenarios [5, 24, 29, 39, 40, 42, 45].

Recently, cross-modal vector search has garnered significant interest alongside the development of multimodal data representation techniques such as CLIP [36]. These models encode data from different modalities, such as vision and natural language, into a shared high-dimensional space, making cross-modal similarities semantically meaningful. Consequently, a query from one modality, such as text, can retrieve semantically relevant objects from another modality, such as images or videos, through ANN search [16, 21, 28]. However, semantic alignment in a shared space does not require the modality-conditioned embeddings to follow identical distributions or to induce identical local-neighbor structures. Even when image and text encoders are jointly trained, differences in the information carried by the two modalities can preserve a modality gap and organize their embeddings differently in the shared space [26, 37]. Accordingly, out-of-distribution (OOD) in this paper is defined relative to the distribution represented by the base dataset and its index. Queries drawn from a distribution different from that represented by the base dataset are termed OOD queries, whereas queries drawn from the same distribution are termed in-distribution (ID) queries. Beyond cross-modal vector search, OOD queries are encountered in various scenarios. For instance, in Transformer-based Large Language Models, query vectors and key-value vectors in the attention mechanism also manifest as OOD [27]. Thus, given the diverse and critical application scenarios, there is a substantial demand for effective ANN searches on OOD queries.

Recent studies [4, 14, 18] have shown that PGs suffer a significant accuracy drop on OOD queries. For instance, as indicated in Table 1, HNSW [31], one of the state-of-the-art methods for processing ID queries, requires over 20x more time to achieve the same quality, e.g., $recall@10 = 0.95$, on OOD queries. To address this issue, existing approaches propose incorporating knowledge from OOD queries in proximity graph construction to make the

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 21, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

index OOD-compatible. Specifically, they all need a historical OOD query set H . RobustVamana [18] builds a proximity graph using the combined set $X \cup H$ for OOD queries, using nodes in H for navigation during search while excluding them from the final results. To enhance search efficiency by avoiding traversing nodes in H , RoarGraph [4] first builds a graph that adds edges among the exact k -ANN (i.e., k -NN) in X , for each $q \in H$. It then enhances graph connectivity by assigning more neighbors to each node via treating it as a query point to search on the graph. NGFix [14] utilizes H in a reverse order: it first builds a proximity graph for ID queries, e.g., HNSW, then adjusts certain edges so that the k -NN of each $q \in H$ becomes accessible during search. Although these approaches effectively enhance the search performance for OOD queries, they still have issues. (1) In real-world scenarios, queries often exhibit a blend of ID and OOD types. Existing methods rely on Mahalanobis distance for distinction, which incurs substantial additional costs. Even after query types are distinguished, without other strategies, the indexes constructed in existing methods diminish performance when handling ID queries, shown in Table 1, as adjustments are made to the initially built proximity graph. (2) They require the same number of historical OOD queries as the base dataset, i.e., $|X| = |H|$, which is impractical. Firstly, scarcity may limit the availability of historical queries for index construction. Secondly, they overlook the method for processing these historical OOD queries when they are encountered beforehand. Using the index built only on X requires substantial time to yield high-quality results, while waiting to accumulate a certain number of OOD queries before constructing their index and then processing the queries leads to prolonging the waiting time for users. (3) RoarGraph and NGFix require exact k -ANN results for vectors in H to construct the index, a time-consuming process. Even with the utilization of an A100 GPU for the computation of these exact k -ANN, as shown in Table 1, the time required, indicated in the index time cost parentheses, still amounts to multiple hours.

In this paper, we propose a prior-free index, PFIndex, for processing both ID and OOD queries in the best way, where prior-free refers to the fact that no pre-collected OOD query set is required before deployment. The main contributions of this work are summarized below. ❶ We design a two-phase search algorithm where the results from the first phase can distinguish between ID and OOD queries. Next, the algorithm provides two separate search strategies for ID and OOD queries. The search in the second phase for both ID and OOD queries is continuous and builds upon the first phase, such that the differentiation process is seamlessly integrated into the search without adding extra time. Moreover, we guarantee that the combined effect of the two phases is the same as beam search on the proximity graph for ID queries, thus maintaining optimal search performance, as shown in Table 1. ❷ We introduce a prior-free iterative strategy to eliminate the utilization of H . By iteratively utilizing the results of OOD queries to construct the augmented proximity graph, the number of nodes traversed in the second phase of our search can gradually be decreased to enhance search efficiency. ❸ We propose an augmented proximity graph for supporting the iterative strategy, which facilitates incremental edge additions to progressively enhance support for OOD queries. Furthermore, it preserves all edges in the proximity graph G constructed for X to support ID queries. Table 1 shows the augmented

proximity graph in PFIndex incurs only a marginal extra cost by storing differential edges derived from G , with the data in parentheses indicating HNSW size. ❹ We present an efficient algorithm for selecting added edges in constructing the augmented proximity graph without exact k -NN, enabling the simultaneous processing of queries and index construction. Table 1 shows that the total time required for PFIndex in query processing and index construction is significantly reduced compared to the time taken by existing methods solely for index construction. ❺ Comprehensive experiments on real-world datasets show that PFIndex outperforms existing state-of-the-art approaches on OOD queries while maintaining the best performance on ID queries. It also performs favorably under mixed and online ID/OOD workloads. For deployments that jointly support ID and OOD queries, PFIndex reduces the total index memory by up to 36.7% and achieves up to 3.18x faster index construction than the baselines, even when an A100 GPU is used to compute their required k -NN results.

The article is organized as follows. We give the preliminaries in Section 2, and discuss the issues of existing approaches in Section 3. We propose a new prior-free index to support both ID and OOD queries in Section 4, and design a method to distinguish the ID and OOD queries in Section 5. In Section 6, we present our search algorithm and the iterative processing strategy. We conduct extensive experiments to confirm the effectiveness and efficiency of our approach in Section 7, and conclude this work in Section 8.

2 Preliminary

In this paper, we study the vector search query in vector databases. **k -ANN Queries:** 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 $\mathbb{R}^d$. The distance between two vectors u and v in $\mathbb{R}^d$ is denoted by $\delta(u, v)$, typically calculated using metrics such as cosine distance, L_2 norm distance, or inner product. In this paper, for simplicity, for a vector q , we assume that $\delta(q, u) < \delta(q, v)$ means vector u is closer to q than v . A vector search query (k -ANN query, or simply a query) gives a query vector $q \in \mathbb{R}^d$, it aims to find the k -approximate nearest neighbors (k -ANN) in X for the user-given query vector q . We use $recall@k$ to measure the quality of a k -ANN query q results, which is defined as $|X_G \cap X_R|/k$, where X_G is the ground-truth k -nearest vector set of q , and X_R is the set found by a k -ANN algorithm for q .

Out-of-Distribution Vector Queries: In cross-modal vector search scenarios, the vectors in the base dataset X and the query vectors can originate from distinct modal objects. For instance, in tasks like image-text retrieval, image and text embeddings are generated by separate encoders. Next, these embeddings are aligned into a common space through techniques such as contrastive learning, exemplified by CLIP [36]. Nevertheless, a modality gap still persists between image and text vectors, leading to differing distributions. According to the existing studies [4, 14, 18], to quantify these distribution disparities, the Wasserstein distance [20] is employed to mathematically measure the distinctions between the two distributions, and the Mahalanobis distance [30] is utilized to measure the distance from a vector to a distribution. A vector q is considered out-of-distribution if its Mahalanobis distance $D_M(q, X)$ to the base dataset X significantly differs from the distances between base vectors [4, 18], e.g., $D_M(x_i, X)$ for $x_i \in X$.

Algorithm 1: BeamSearch (G, q, k, w)

Input : PG G , query vector q , k for top- k , and beam width w
Output : k -ANN of q

```

1  $S \leftarrow \{\text{the entry node of } G\};$ 
2 while there exists an unvisited node in  $S$  do
3    $u \leftarrow$  the nearest unvisited node to  $q$  in  $S$ ;
4   Mark  $u$  visited;
5   for each unvisited  $v$  of  $(u, v) \in G$  that is not in  $S$  do
6     Insert  $v$  into  $S$ ;
7   while  $|S| > w$  do Remove the farthest nodes to  $q$  from  $S$ ;
8 return the top- $k$  nearest nodes to  $q$  in  $S$ ;
```

In cross-modal ANN search, vectors in dataset X are presumed to pertain to the same modality, implying identical distributions, whereas the query vector q may manifest a distribution divergent from those in X . Queries with different distributions compared to X are termed out-of-distribution (OOD) queries, while those having the same distribution as X are termed in-distribution (ID) queries.

Proximity Graphs: For an ID k -ANN query q , i.e., which aims to find the k -ANN from X , the state-of-the-art approaches, e.g., NSG [9] and HNSW [31], can support it using a proximity graph built. Here, a proximity graph (PG) $G = (V, E)$, where $v_i \in V$ represents a unique vector v_i , and an edge $(v_i, v_j) \in E$ represents that two vectors are close to each other based on the distance function $\delta(v_i, v_j)$, or precisely v_j is one of the k -nearest neighbors of v_i . Proximity graphs feature an out-degree constraint denoted as M , a hyperparameter specified during index construction. The RNG pruning strategy is commonly employed in various proximity graphs, e.g., [9, 12, 31], to restrict the out-degree of each node $u \in G$. Specifically, an edge (u, v) is included in the proximity graph only if there exists no edge (u, w) such that $\delta(u, w) < \delta(u, v)$ and $\delta(v, w) < \delta(u, v)$. Following the utilization of RNG pruning strategy, the farthest node v from u , i.e., u 's neighbor v with the largest value $\delta(u, v)$, will be eliminated from the out-neighbor list of u if the out-degree of u remains larger than M . Below, for simplicity, a graph is a proximity graph, and we use x_i and v_i interchangeably, as a node v_i in G represents a point x_i in X .

Beam Search: Over a proximity graph, G , constructed, a widely used greedy algorithm is presented in Algorithm 1 to find the k -ANN for an ID query vector q [8, 9, 12, 31, 35, 43]. Initially, it starts the search at the entry node of G , and inserts it into set S , which is unvisited (line 1). In the while-loop (lines 2-7), if there exists an unvisited node in S , the beam search tries to find the closer vectors of q in G as follows. First, it selects the nearest unvisited node, u , to the query vector q from S , and marks it visited (lines 3-4). Second, it adds every node, v , into S if there is a directed edge from u to v , and v is not in S which implies that v is unvisited yet (lines 5-6). Third, it deletes the farthest nodes to q from S if $|S| > w$ (line 7). When the algorithm breaks, the results are returned (line 8).

Beam search on proximity graphs is commonly viewed as having two phases [14, 43, 46]. In the first phase, the search moves toward the target region and locates one of the query vector q 's k -NN. In the second phase, starting from some k -NN found in the first phase, the search explores the remaining k -NN by traversing their out-edges in an approximate manner. The first phase relies on the navigability

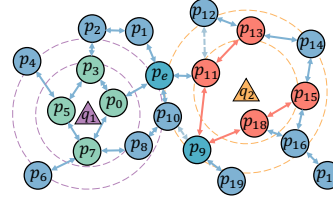


Figure 1: An Example of Proximity Graph and ID/OOD Queries

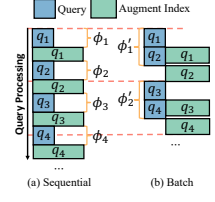


Figure 2: Illustration of Iterative Strategy

properties of the constructed proximity graph [9, 31, 43]. Proximity graphs perform well on ID queries mainly because of the second phase: if a vector v is close to q , then vectors near v are also likely to be close to q . Accordingly, proximity graphs connect nearby vectors to facilitate efficient search. For example, in Fig. 1, consider q_1 as an ID query with $\{p_5, p_0, p_7, p_3\}$ as its 4-NN. The interconnectedness of q_1 's 4-NN in the graph illustrates the ease of locating all neighbors by finding one of them. However, existing analyses [4, 14] highlight the limitations of such property when dealing with OOD queries.

Example 2.1: Fig. 1 shows an example of an OOD query q_2 , where the edges in the proximity graph, built on dataset X , are colored in blue. Here, p_e is the entry point of the proximity graph. The search starts at p_e and locates p_{11} as one of the 4-NN of q_2 . However, as all its neighbors are further from q_2 , the search halts. Thus, the remaining 4-NN of q cannot be discovered due to the lack of connection. Furthermore, if node p_9 is the entry point, the search will promptly cease. In this case, with no closer neighbors to q_2 , none of the 4-NN will be discovered.

3 The Issues of Existing Approaches

Existing studies have identified that proximity graphs constructed for the dataset X may not be effectively used to answer OOD queries. This limitation arises from the fact that proximity graphs are built by solely considering ID vectors. For example, Fig. 3 presents a scenario where the vectors $x_1, \dots, x_9$ in X are all near the hyperball's boundary (depicted by the dashed lines), while the OOD query vector q is positioned differently within the distribution compared to the vectors in X , at around the center of a hyperball. The proximity of vectors in the distribution of X , i.e., the edges in the proximity graph built on X , is along the hyperball's boundary, e.g., x_1 is close to x_5 and x_9 . Thus, the 4-NN of q , x_1, x_2, x_3, x_4 (colored in red), are disconnected in the proximity graph, edges colored blue, due to their discontinuity in the distribution. Hence, even if one of q 's 4-NN is identified, finding all of them is still difficult.

To address this issue, current approaches propose the inclusion of a set H containing historical OOD query vectors during the proximity graph construction. By involving the vectors in H , the built proximity graphs do not exclusively focus on a single modality, thereby enhancing their suitability for handling OOD queries.

RobustVamana [18]: RobustVamana proposes a simple but effective solution: it directly builds a proximity graph based on the set $X \cup H$, allowing navigation using $q \in H$ during queries, but excludes them from the final results. For example, in Fig. 3, q is included in the proximity graph for search, but is not included in the results.

Issue: A contradiction arises within this solution. On one hand, an ample quantity of OOD query vectors is required to participate

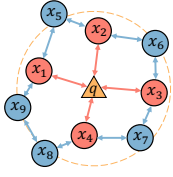


Figure 3: Illustration of OOD Query

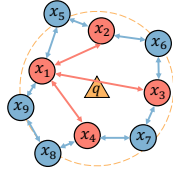


Figure 4: An Example of RoarGraph

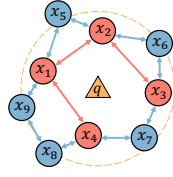


Figure 5: An Example of NGFix

in proximity graph construction to enhance the graph’s efficacy for OOD query vectors. Conversely, if numerous OOD queries are included in the proximity graph, the beam search traversing the graph will encounter many OOD query vectors, necessitating distance computations for navigation. Yet, such computations are a waste since these vectors are ultimately excluded from the final results, leading to a subsequent decline in performance.

RoarGraph [4]: RoarGraph utilizes projection operations to circumvent the direct addition of vectors $q \in H$ into the graph index, for addressing the contradiction in RobustVamana. Initially, RoarGraph constructs a bipartite graph: one set of nodes corresponds to vectors in X , while the other set represents OOD vectors in H . Instead of establishing edges between $x_i \in X$ and $q \in H$ in the proximity graph, RoarGraph first builds a projected graph by linking p_1 with each of $p_2, \dots, p_{N_q}$ for each OOD vector $q \in H$, where p_i represents q ’s i -nearest neighbor in X and N_q is a predetermined hyperparameter. For example, x_1 connects to x_2, x_3, x_4 as shown in Fig. 4. Next, RoarGraph treats each vector in X as a query vector, executing a k -ANN search on the projected graph to assign more neighbors to each node for enhancing the graph connectivity.

NGFix [14]: NGFix introduces Escape Hardness to assess the difficulty of greedy searches from a node u to another node v on the proximity graph G for a query vector q , denoted as $EH(u, v, q, G)$. A higher value of $EH(u, v, q, G)$ indicates that v is more difficult to reach from u during the greedy search on G for query q . NGFix then uses OOD query vectors $q \in H$ to diminish the Escape Hardness among their k -nearest neighbors, denoted as $K(q) = \{p_1, p_2, \dots, p_k\}$. Specifically, NGFix first builds a proximity graph G , e.g., HNSW. Next, for a vector $q \in H$, NGFix computes Escape Hardness between all pairs of its k -nearest neighbors in $O(k^3)$. It then introduces a threshold K_h and treats the directed edge $(p_i, p_j) \notin G$, where $p_i, p_j \in K(q)$, as a candidate for insertion when $EH(p_i, p_j, q, G) > K_h$. Adding (p_i, p_j) reduces the difficulty of reaching p_j from p_i during greedy search for q . Given that a large number of such edges may exist, NGFix selects a maximum of $k - 1$ edges adding to G among the candidate edges to (1) ensure all nodes in $K(q)$ are connected and (2) minimize the weight of the chosen edges, where the weight of an edge (p_i, p_j) corresponds to their distance $\delta(p_i, p_j)$. For example, edges (x_1, x_2) , (x_1, x_4) and (x_2, x_3) are added in Fig. 5. Moreover, NGFix adds certain edges when the greedy search on G fails to reach any nodes in $K(p)$, enabling nodes in $K(p)$ to be accessed through a greedy search from the entry point in G .

Although the performance for OOD query vectors on the existing three indexes has been improved by utilizing H , they still encounter several issues as discussed below.

In-Distribution Issue: In real-world scenarios, queries may not consistently fit into the categories of ID or OOD; they often present

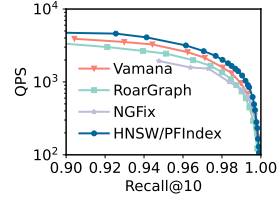


Figure 6: Search Performance for ID Queries on LAION

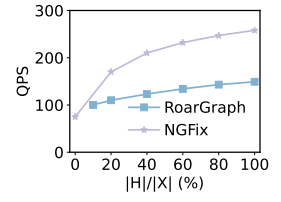


Figure 7: Search Performance v.s. $|H|/|X|$

in a mixed style. Although Mahalanobis distance offers a way to differentiate query types, computing it adds extra overhead to each query, which compromises the search efficiency.

Moreover, even after distinguishing the query type, the existing methods still show compromised performance for ID queries, since they primarily focus on designing a proximity graph index for OOD queries. As illustrated in Fig. 6, both RoarGraph and NGFix exhibit diminished search efficacy when compared to HNSW, which is exclusively built for the vectors in X .

Query Vector Issue: The major issue of existing approaches is the set H of historical OOD query vectors utilized in index construction. The existing three methods all assume prior knowledge of the OOD query vectors, and their indexes rely on a significant size of H during graph construction to achieve satisfactory performance, i.e., $|H| = |X|$. Fig. 7 illustrates the query per second (QPS) at $recall@100 = 0.98$ on LAION dataset for RoarGraph and NGFix with varying numbers of OOD query vectors, i.e., the size of H , utilized in index construction. The LAION base dataset contains 10 million vectors, which means the size of H required in their approaches is on the same scale as the size of the base dataset X , posing a major practical issue.

Firstly, complete knowledge of OOD queries may not be available due to the scarcity of historical OOD queries. Secondly, they overlook the method for processing these historical OOD queries when they are encountered beforehand. For users who make the initial OOD queries, they need to endure prolonged wait times to obtain high-quality results for an effective index that accumulates a large number of OOD queries before construction. Another option is to search on the proximity graph built on X , which may yield suboptimal results or high latency for superior quality results.

Index Cost Issue: Both RoarGraph and NGFix grapple with substantial index construction costs. A primary cost arises from the necessity of exact or extremely high-quality k -ANN (i.e., k -NN) for each OOD query vector during index construction. As discussed, since they require an equivalent number of OOD query vectors as the dataset size, this results in a theoretical time complexity of $O(n^2d)$ for computing ground-truth k -NN, a complexity deemed impractical in existing studies [6, 9, 35, 43]. As shown in Table 1, even with the utilization of an A100 GPU, computing k -NN for H in LAION dataset, for instance, needs 3.97 hours. This duration exceeds the time needed to build a proximity graph on X , for instance, constructing an HNSW on LAION dataset requires 1.19 hours. Furthermore, for NGFix, its Escape Hardness computation among $K(q)$ for each $q \in H$ is the second main cost, since $|K(q)|$ is always set to 500 [14], and computation needs $O(|K(q)|^3)$.

4 A Prior-Free All-in-One Approach

In this section, we propose a prior-free all-in-one index to deal with the three issues, namely, the in-distribution issue, the query vector issue, and the index cost issue, and then give the details.

In-Distribution Issue: To address the in-distribution issue, the first step is to distinguish between ID and OOD queries in a light-weight manner. To achieve this, we design a *two-phase search algorithm*. The first phase searches the original proximity graph constructed for the base dataset X and obtains the k -th approximate nearest node to query q , denoted as $\mathbf{p}_k(q)$. Next, we utilize the normalized distance $\delta(\mathbf{p}_k(q), q)$ to discern the query type; the details will be presented in Section 6.1. After the query type is determined, the second phase adopts a type-specific graph view. (1) For an ID query, the search continues on the original proximity graph G constructed for X . We prove that the combined execution of the two phases is equivalent to beam search on G , thereby preserving the search performance for ID queries. (2) For an OOD query, instead of searching on G , we conduct the search on the augmented proximity graph G^+ specifically built for them as defined below.

Augmented Proximity Graph: The augmented proximity graph G^+ comprises three types of edges: (1) ID edges, existing in G constructed from dataset X ; (2) OOD edges, added to enhance performance for OOD queries; and (3) $\neg$ ID edges, existing in G built for X but are pruned due to the new additions of OOD edges.

Hence, in the second phase of search for OOD queries, we use the edges labeled ID and OOD. The edges labeled OOD are added to enhance the search performance for OOD queries. Details will be provided in this section later. It is important to note that the graph G does not need to be stored, as the ID and $\neg$ ID edges in G^+ are precisely the proximity graph G constructed on X .

Query Vector Issue: To address this issue, we introduce a prior-free method that operates without prior knowledge of the OOD query vectors, which processes incoming OOD queries and augments the graph iteratively. For an OOD query, we obtain its k -ANN results with the parameter ϕ , and then employ these results to adjust the graph G^+ for enhanced OOD query support. The parameter ϕ is used in the second phase as a lower bound on the number of nodes whose distances to the query must have been evaluated before the search is allowed to terminate. Thus, a larger ϕ forces a more extensive search and generally produces higher-quality results for augmenting the graph. Since the processed OOD queries contribute their results to graph augmentation, the searchable regions become progressively better connected, allowing the subsequent queries to use a smaller ϕ . For instance, as illustrated in Fig. 2(a), we process the OOD queries $q_1, \dots, q_4$ sequentially, where the k -ANN results from each query are utilized to augment the index for the subsequent query, and so forth, resulting in a reduction of ϕ required for each query, i.e., $\phi_1 > \phi_2 > \dots$ in Fig. 2. Further details on this iterative strategy will be presented in Section 6.2.

Index Cost Issue: We eliminate the separate pre-deployment exact-kNN stage over a large historical OOD set through the iterative strategy. It is because it can consistently provide high-quality k -ANN results for constructing the index, eliminating the necessity for obtaining ground-truth k -NN. Furthermore, we introduce an efficient method for constructing the augmented proximity graph

to enhance the support for OOD queries, thereby reducing index construction time.

As analyzed in Section 2, the two issues in the proximity graph for OOD queries exist in two phases of the beam search: (1) locating one of the k -NN of an OOD query q from the entry point and (2) finding all k -NN of an OOD query q from any other k -NN.

As discussed, NGFix proposes a method based on Escape Hardness to address these issues, aiming to reduce the difficulty of navigating from one k -NN to another k -NN by introducing additional edges. While effective, this approach incurs high costs in computing the hardness in $O(k^3)$ and determining which edges to add in $O(d \cdot k^2)$. We propose a direct method to address issues based on the following fact.

Fact 4.1: Given a query q , let $p_1, \dots, p_k$ denote its k -NN and $\text{rank}(p_i)$ be the rank of p_i for q , i.e., $\text{rank}(p_i) = i$. When beam width $w \geq k$ in Algorithm 1, p_k can be found by p_i if there exists a path $[x_1, \dots, x_\mu]$ from p_i to p_k satisfying $x_1 = p_i, x_\mu = p_k$ and $\forall j \in \{1, 2, \dots, \mu - 1\}, \text{rank}(x_j) < \text{rank}(x_{j+1})$.

Based on Fact 4.1, let $\mathcal{N}(q) = \{p_1, p_2, \dots, p_k\}$ represent the set of k -NN for a query q . We can incrementally add edges between p_j and p_i based on the ascending order of j value, where $i < j$. In Algorithm 1, when searching p_j for q , we always have $w \geq j$, allowing p_j to be navigated once any of p_i (where $i < j$) is found.

Next, we address the first issue, where none of $p_i \in \mathcal{N}(q)$ is found. In this case, we consider adding edges from the currently found nodes to nodes in $\mathcal{N}(q)$. Specifically, let $\mathbf{p}_1(q)$ be 1-ANN obtained from the proximity graph built for X in the first phase, adding edges from $\mathbf{p}_1(q)$ to nodes in $\mathcal{N}(q)$ will facilitate search in finding nodes in $\mathcal{N}(q)$. For example, referring back to Example 2.1, we have $\mathbf{p}_1(q) = p_9$ and $\mathcal{N}(q) = \{p_{11}, p_{13}, p_{15}, p_{18}\}$ when the query is q_2 with the entry point p_9 . We add edges from p_9 to p_{11} and p_{18} to address the first issue, such that the search starting at p_9 can navigate to nodes in $\mathcal{N}(q)$. Next, edges (p_{11}, p_{13}) and (p_{18}, p_{15}) will be added since $\text{rank}(p_{11}) < \text{rank}(p_{13})$ and $\text{rank}(p_{18}) < \text{rank}(p_{15})$.

Query-Local Degree Limit B_q : To prevent edge concentration on a single node, we introduce a parameter B_q to restrict the degree of each node. This parameter is selected according to the capacity of an ideal three-level bounded-degree tree. With currently $k+1$ nodes ($\mathbf{p}_1(q)$ and nodes in $\mathcal{N}(q)$), at most B_q edges are added from $\mathbf{p}_1(q)$ to nodes in $\mathcal{N}(q)$, denoted as $\mathcal{N}^1(q)$, hence, $|\mathcal{N}^1(q)| = B_q$. Nodes in $\mathcal{N}^1(q)$ already have an edge to $\mathbf{p}_1(q)$, allowing $B_q(B_q - 1)$ nodes to be reached from nodes in $\mathcal{N}^1(q)$, denoted as $\mathcal{N}^2(q)$, satisfying $|\mathcal{N}^2(q)| = B_q(B_q - 1)$. Hence, the remaining nodes are expected to be reachable from $\mathcal{N}^2(q)$, designated as $\mathcal{N}^3(q)$, with $|\mathcal{N}^3(q)| = |\mathcal{N}^2(q)|(B_q - 1)$. Thus, we can determine B_q as the minimum integer value such that $|\mathcal{N}^1(q)| + |\mathcal{N}^2(q)| + |\mathcal{N}^3(q)| = B_q^3 - B_q^2 + B_q \geq k$. For instance, setting k as 100 needs $B_q = 5$.

To ensure a natural addition of edges in the proximity graph, we prioritize the addition of short-distance edges. In Algorithm 2, the process begins with the computation of the degree limit B_q (line 1). It starts adding edges from $\mathbf{p}_1(q)$ by adding $\mathbf{p}_1(q)$ to the candidate set C , and the edge set $\mathcal{E}$ is initialized as empty (line 2). If $\mathbf{p}_1(q)$ is included in $\mathcal{N}(q)$, it is removed from $\mathcal{N}(q)$ (line 3). The algorithm then progresses through each node in $\mathcal{N}(q)$ in ascending order of the rank for q (line 4). For every node u , we add an edge between it

Algorithm 2: EnhanceConnectivity ($q, \mathbf{p}_1(q), \mathcal{N}(q)$)

Input : OOD query vector $q, \mathbf{p}_1(q), \mathcal{N}(q)$ for q
Output : edge set $\mathcal{E}$ to enhance connectivity

- 1 $B_q \leftarrow \min\{B_q \in \mathbb{Z}^+ \mid B_q^3 - B_q^2 + B_q \geq |\mathcal{N}(q)|\};$
- 2 $\mathcal{E} \leftarrow \emptyset; C \leftarrow \{\mathbf{p}_1(q)\};$
- 3 Remove $\mathbf{p}_1(q)$ from $\mathcal{N}(q)$ if $\mathbf{p}_1(q) \in \mathcal{N}(q)$;
- 4 **for** each $u \in \mathcal{N}(q)$ with ascending order of $\text{rank}(u)$ **do**
- 5 $v \leftarrow$ the node in C with smallest distance $\delta(u, v)$;
- 6 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(u, v), (v, u)\};$
- 7 **if** v has been added B_q edges **then** $C \leftarrow C \setminus \{v\};$
- 8 $C \leftarrow C \cup \{u\};$
- 9 **return** $\mathcal{E};$

and the node v with a lower rank that exhibits the smallest distance between u and v (lines 5-6). If the out-degree of v reaches the limit B_q , we eliminate it from the candidate set C (line 7), and next include u in the candidate set C for future edge additions (line 8). Notably, in Algorithm 2, if $\mathbf{p}_1(q) \in \mathcal{N}(q)$, this process remains viable as it becomes part of our goal to address the second issue.

Time Complexity: Note that line 5 can be implemented in two ways. A direct scan over C exactly evaluates all candidates and requires $O(d|C|) = O(dk)$ time per iteration, resulting in $O(dk^2)$ time. Alternatively, when k is set to a large value to involve more neighbors and improve graph-augmentation quality, we can build an HNSW. Specifically, for each u at line 4, we obtain its 1-ANN via k -ANN search on HNSW built for C and next add it and edges between it and its k -ANN to HNSW at line 8. In line 7, we do not remove v from HNSW; instead, we flag it and exclude it from the search results. Hence, the overall complexity of Algorithm 2 is approximately $O(d \cdot k \ln k)$, when assuming the search on HNSW is empirically $O(d \cdot \ln k)$ [35, 43].

The acquisition of $\mathbf{p}_1(q)$ and $\mathcal{N}(q)$ will be detailed in Section 6, a process facilitated during our two-phase search algorithm.

Next, we can present the process of building an augmented proximity graph, augmenting the proximity graph by each OOD query q . As shown in Algorithm 3, for each OOD query q , Algorithm 2 is initially invoked to generate a candidate edge set (line 1). Next, each edge is considered for addition (line 2). If an edge is already labeled as ID or OOD, it is skipped (line 3). If it is labeled $\neg$ ID, it is relabeled as ID for future pruning (line 4). If the edge is not included in the proximity graph, it is labeled as OOD (line 5). In cases where the number of OOD edges surpasses M (line 6), the RNG pruning strategy is employed to ensure the OOD edges do not exceed this limit (line 7). Alternatively, if the number of OOD edges is within the limit but the total number of edges labeled OOD or ID exceed M (line 8), an ID edge is selected using the RNG pruning strategy, and its label is set to $\neg$ ID (line 9). For instance, in Fig. 1, the OOD edges are colored in red. When $M = 3$, the edges (p_{11}, p_{12}) and (p_9, p_{10}) will be labeled as $\neg$ ID through the utilization of the RNG pruning strategy, represented as dashed lines in Fig. 1.

5 OOD Queries Distinguishment

In this section, we focus on developing a lightweight method to distinguish OOD queries for addressing the in-distribution issue. Here, we do not intend to identify the semantic modality or provide

Algorithm 3: AugmentPG ($G^+, q, \mathbf{p}_1(q), \mathcal{N}(q), M$)

Input : augmented PG G^+ , OOD query vector $q, \mathbf{p}_1(q), \mathcal{N}(q)$ for q , and out-degree limit M
Output : a new augmented PG G^+

- 1 $\mathcal{E} \leftarrow \text{EnhanceConnectivity}(q, \mathbf{p}_1(q), \mathcal{N}(q));$
- 2 **for** each edge $(u, v) \in \mathcal{E}$ **do**
- 3 **if** $(u, v) \in G^+$ with label ID or OOD **then** continue;
- 4 **else if** $(u, v) \in G^+$ with label $\neg$ ID **then** label edge (u, v) ID;
- 5 **else** add edge (u, v) to G^+ with label OOD;
- 6 **if** $M <$ the number of edges from u labeled OOD **then**
- 7 Remove an edge from u labeled OOD by RNG pruning strategy as defined in Section 2;
- 8 **else if** $M <$ the number of edges from u labeled OOD or ID **then**
- 9 Select an edge from u labeled ID by RNG pruning strategy as defined in Section 2, and relabel this edge $\neg$ ID;
- 10 **return** $G^+;$

a universal statistical test for OOD detection. Instead, we estimate whether a query is likely to be difficult for the proximity graph constructed on the base dataset X . Queries that appear sufficiently close to dense, well-connected base-data regions continue searching on the original graph, whereas queries that appear to lie in sparse or weakly connected regions use the OOD-oriented graph view.

Existing approaches [4, 14, 18] employ Mahalanobis distance [30] to discern whether the query vector is OOD with respect to the dataset vectors. For a dataset $X = \{x_1, \dots, x_n\}$, let $\mu_X = \frac{1}{n} \sum_{i=1}^n x_i$ be the mean vector and $\sigma_X = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu_X)(x_i - \mu_X)^T$ be the covariance matrix. Mahalanobis distance between a query q and dataset X is defined as $D_M(q, X) = \sqrt{(q - \mu_X)^T \sigma_X^{-1} (q - \mu_X)}$. Computation of μ_X , σ_X , and σ_X^{-1} entails $O(nd^2 + d^3)$ time, with an additional $O(d^2)$ time for each query to calculate Mahalanobis distance. However, an issue arises. According to studies [9, 31, 35, 43], the k -ANN searches on a proximity graph empirically need $O(d \cdot \ln n)$, while the Mahalanobis distance computation needs $O(d^2)$ time. The time for Mahalanobis distance is approaching or even surpassing the time needed for the entire search. Moreover, the preprocessing necessary for the Mahalanobis distance is akin to the time taken for proximity graph index construction, which approximates $O(nd \ln n)$. This underscores the need for a more lightweight approach to distinguish if a query vector is OOD.

Here, we introduce a simple yet effective strategy for distinguishing OOD queries, followed by a theoretical analysis.

k -NN Normalized Distance: For every nonzero vector x , let $\bar{x} = x/\|x\|_2$ denote its L_2 -normalized representation. For a query q , the k -NN normalized distance refers to the normalized distance between the query vector q and its k -NN in X . Formally, for a query vector q , its k -NN in X is denoted as $\mathbf{p}_k(q)$, and the distance between the normalized query vector $\bar{q}$ and the normalized vector $\bar{\mathbf{p}}_k(q)$ is represented as $\bar{\delta}(\mathbf{p}_k(q), q) = \delta(\bar{q}, \bar{\mathbf{p}}_k(q))$. Since Euclidean distance, cosine distance, and inner-product similarity are equivalent up to monotonic transformations for unit-normalized vectors, we use $\bar{\delta}(\mathbf{p}_k(q), q) = -\bar{q}^T \bar{\mathbf{p}}_k(q)$ as the k -NN normalized distance.

The k -NN normalized distance measures the radius of the local base-data neighbor required to contain k vectors. A small $\bar{\delta}(\mathbf{p}_k(q), q)$

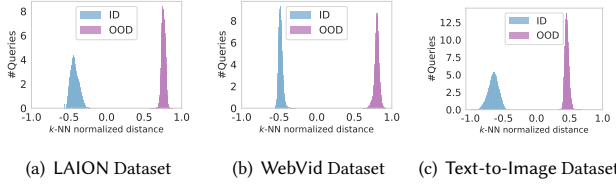


Figure 8: k -NN Normalized Distances $\bar{\delta}(\mathbf{p}_k(q), q) = -\bar{q}^\top \bar{\mathbf{p}}_k(q)$

indicates that q lies in a dense base-data region, where the original proximity graph is likely to provide sufficient local connectivity. A large $\bar{\delta}(\mathbf{p}_k(q), q)$ indicates that q lies in a relatively sparse base-data region, where the original graph is more likely to exhibit the OOD search issues discussed in Section 3. If an OOD query nevertheless lies in a dense and well-connected base-data region, $\bar{\delta}(\mathbf{p}_k(q), q)$ can remain small. Such a query may be routed as ID, but this does not create a significant problem: the original graph is already suitable for processing it. Therefore, we employ a threshold-based criterion to distinguish between ID and OOD queries. Specifically, we classify a query q as an OOD query if $\bar{\delta}(\mathbf{p}_k(q), q) \geq \tau$, where τ is set to $\max_{x_i \in X} (\bar{\delta}(\mathbf{p}_k(x_i), x_i))$. Given our assumption of a single modular dataset in X , all representing in-distribution vectors. Fig. 8 illustrates the k -NN normalized distances of both ID and OOD queries. The results show a clear distinction, with significantly higher k -NN normalized distances for OOD queries compared to ID queries, showcasing a distinct boundary that simplifies the differentiation between the two.

One issue remains: obtaining the exact $\mathbf{p}_k(q)$ is expensive. Notably, the proximity graph built on X may not adequately support OOD queries, so the k -ANN results returned in the first phase can be of lower quality than those for ID queries. Thus, instead of using the exact k -NN distance, we use the first-phase result, denoted by $\mathbf{p}_k(q)$, for distinction. In this case, $\bar{\delta}(\mathbf{p}_k(q), q) \geq \bar{\delta}(\mathbf{p}_k(q), q)$ holds, and due to the lower quality of results, potentially magnifying the gap. Hence, utilizing $\bar{\delta}(\mathbf{p}_k(q), q)$ for OOD query classification may be further easier. To further avoid computing exact k -NN for threshold selection, we define the default threshold as $\tau_0 = \max_{x_i \in X} \bar{\delta}(\mathbf{p}'_k(x_i), x_i)$, where $\mathbf{p}'_k(x_i)$ denotes the k -th nearest out-neighbor of x_i in the proximity graph, which is a high-quality k -ANN of x_i . Since all $x_i \in X$ are ID vectors, τ_0 serves as an empirical upper bound of the normalized k -NN distance for ID queries. As shown in Fig. 8, the normalized k -NN distance yields a clear separation between ID and OOD queries; on our current datasets, we observe 100% distinction accuracy when $\tau = \tau_0$, and we further study the sensitivity to this threshold in Section 7 (Exp. 8).

For the following theoretical analysis, we distinguish the geometric k -NN radius from the inner-product score used before. Let $\bar{\mathbf{p}}_{k,n}(\bar{q})$ denote the exact k -th nearest neighbor of $\bar{q}$ in the normalized base dataset $\bar{X}_n = \{\bar{x}_1, \dots, \bar{x}_n\}$, under the ambient Euclidean distance, and define $R_{k,n}(\bar{q}) = \|\bar{q} - \bar{\mathbf{p}}_{k,n}(\bar{q})\|_2$. For unit-normalized vectors, this radius is monotonically equivalent to the exact inner-product score, since $R_{k,n}(\bar{q})^2 = 2 - 2\bar{q}^\top \bar{\mathbf{p}}_{k,n}(\bar{q})$.

Theorem 5.1: Let $\mathcal{M} \subseteq \mathbb{S}^{d-1}$ be a C^2 smooth m -dimensional manifold embedded in $\mathbb{R}^d$, and let $\bar{q} \in \mathcal{M}$ be a fixed interior point. Let $\bar{X}_n = \{\bar{x}_1, \dots, \bar{x}_n\} \subseteq \mathcal{M}$ consist of vectors drawn independently from a probability density f_{in} with respect to the volume measure on $\mathcal{M}$. Assume that $f_{\text{in}}(\bar{q}) > 0$ and that f_{in} is locally Lipschitz-continuous

in a neighbor of $\bar{q}$. Let $k = k(n)$ satisfy $k \rightarrow \infty$ and $\frac{k}{n} \rightarrow 0$ as $n \rightarrow \infty$. Then $\mathbb{E}[R_{k,n}(\bar{q})] = \left(\frac{k}{n V_m f_{\text{in}}(\bar{q})}\right)^{1/m} (1 + o(1))$, where V_m is the volume of the m -dimensional unit ball. Therefore, the expected normalized-space k -NN radius is asymptotically inversely proportional to $f_{\text{in}}(\bar{q})^{1/m}$.

Proof Sketch: Let $F_{\bar{q}}(r) = \Pr(\|\bar{x} - \bar{q}\|_2 \leq r) = \int_{\mathcal{M} \cap B_{\mathbb{R}^d}(\bar{q}, r)} f_{\text{in}}(x) d \text{vol}_{\mathcal{M}}(x)$, where $\bar{x}$ is drawn from f_{in} . Because $\mathcal{M}$ is a smooth m -dimensional manifold and $\bar{q}$ is an interior point, $\text{vol}_{\mathcal{M}}(\mathcal{M} \cap B_{\mathbb{R}^d}(\bar{q}, r)) = V_m r^m (1 + o(1))$ as $r \rightarrow 0$. Together with the local Lipschitz continuity of f_{in} , this gives $F_{\bar{q}}(r) = V_m f_{\text{in}}(\bar{q}) r^m (1 + o(1))$. Therefore, $F_{\bar{q}}^{-1}(u) = \left(\frac{u}{V_m f_{\text{in}}(\bar{q})}\right)^{1/m} (1 + o(1))$ as $u \rightarrow 0$. For each $\bar{x}_i \in \bar{X}_n$, let $D_i = \|\bar{x}_i - \bar{q}\|_2$ and $U_i = F_{\bar{q}}(D_i)$. By the probability integral transform, the U_i are independently uniform on $[0, 1]$. If $U_{(k)}$ denotes their k -th order statistic, then $R_{k,n}(\bar{q}) = F_{\bar{q}}^{-1}(U_{(k)})$, $U_{(k)} \sim \text{Beta}(k, n - k + 1)$. Under $k \rightarrow \infty$ and $k/n \rightarrow 0$, we have $U_{(k)} \rightarrow 0$, and hence $\mathbb{E}[R_{k,n}(\bar{q})] = \left(\frac{1}{V_m f_{\text{in}}(\bar{q})}\right)^{1/m} \mathbb{E}[U_{(k)}^{1/m}] (1 + o(1))$. Using the moment formula of the Beta distribution, $\mathbb{E}[U_{(k)}^{1/m}] = \frac{\Gamma(k + \frac{1}{m}) \Gamma(n+1)}{\Gamma(k) \Gamma(n+1 + \frac{1}{m})} = \left(\frac{k}{n}\right)^{1/m} (1 + o(1))$. $\square$

Theorem 5.1 provides a monotonic local-density interpretation of the normalized-space k -NN radius. In particular, for two points $\bar{q}_1, \bar{q}_2$ satisfying the theorem assumptions, $\frac{\mathbb{E}[R_{k,n}(\bar{q}_1)]}{\mathbb{E}[R_{k,n}(\bar{q}_2)]} = \left(\frac{f_{\text{in}}(\bar{q}_2)}{f_{\text{in}}(\bar{q}_1)}\right)^{1/m} (1 + o(1))$. Thus, a lower local base density corresponds to a larger expected radius, but the exponent $1/m$ compresses density differences as the intrinsic dimension m increases. The numerical strength of the separation therefore depends jointly on the intrinsic dimension and the local-density ratio. Therefore, this distance can serve as a lightweight indicator of whether a query receives weak local support from the base distribution, i.e., the queries that cannot be effectively supported by the base index.

6 Search-Augmentation Strategy

In this section, we introduce an iterative search strategy together with the index augmentation method in Section 4. For consecutive queries, the search algorithm is adjusted dynamically. Initially, OOD queries may incur high search cost because the index is not yet optimized for them; as the index is progressively augmented, we aim to reduce this cost while preserving search quality.

To achieve this, we first propose a two-phase search algorithm with a parameter that controls search cost. We then introduce an iterative method to adjust this parameter dynamically, ensuring high-quality results throughout the augmentation process.

6.1 Two-Phase Search Algorithm

In this section, we propose a two-phase search algorithm designed to support both ID and OOD queries effectively. The two phases have the same goal as the two phases in beam search, as discussed in Section 4, but with different search strategies. **1** In the first phase, we aim to move towards the region containing one of the k -NN of

Algorithm 4: TwoPhaseSearch (G, q, k, w, ϕ)

Input : augmented PG G^+ , query vector q , k for top- k , beam width w , and a parameter ϕ
Output : k -ANN of q

```
1  $V \leftarrow \{\text{the entry node of } G\}; C \leftarrow V;$   
2 while  $|\{v \in V \mid \forall u \in C, \delta(v, q) < \delta(u, q)\}| < k$  do  
3    $v \leftarrow \text{the node in } C \text{ with smallest distance to } q; C \leftarrow C \setminus \{v\};$   
4   for each edge  $(v, u) \in G^+$  labeled ID or  $\neg$ ID do  
5     if  $u \notin V$  then  $C \leftarrow C \cup \{u\}, V \leftarrow V \cup \{u\};$   
6  $\hat{\tau} \leftarrow \bar{\delta}(q, p_k(q))$ , where  $p_k(q)$  is the  $k$ -NN of  $q$  in  $V$ ;  
7 while  $C \neq \emptyset$  do  
8    $v \leftarrow \text{the node in } C \text{ with smallest distance to } q; C \leftarrow C \setminus \{v\};$   
9   if  $\hat{\tau} \leq \tau$  then  
10    for each edge  $(v, u) \in G^+$  labeled ID or  $\neg$ ID do  
11      if  $u \notin V$  then  $C \leftarrow C \cup \{u\}, V \leftarrow V \cup \{u\};$   
12    if  $|\{v \in V \mid \forall u \in C, \delta(v, q) < \delta(u, q)\}| \geq w$  then break;  
13  else  
14    for each edge  $(v, u) \in G^+$  labeled ID or OOD do  
15      if  $u \notin V$  then  $C \leftarrow C \cup \{u\}, V \leftarrow V \cup \{u\};$   
16    if  $|\{v \in V \mid \forall u \in C, \delta(v, q) < \delta(u, q)\}| \geq w$  and  $|V| \geq \phi$  then break;  
17 return the top- $k$  nearest nodes to  $q$  in  $V$ ;
```

a query q . In this phase, we leverage the proximity graphs built for X , i.e., the edges labeled ID and $\neg$ ID from the augmented proximity graph built, such that we do not need to consider adding edges for OOD queries, hence minimizing index cost. ② Based on the result from the first phase, we can distinguish the OOD and ID queries for later different search strategies in the second phase. ③ In the second phase, for ID queries, we use the beam search with beam width w to search, while for OOD queries, we introduce another parameter ϕ to regulate the cost by restricting the traversal of nodes in the proximity graph, i.e., limiting the number of distance computations.

To seamlessly transition from the first phase to the second without restarting the search process, we redesign the S used in beam search in Algorithm 1. We maintain two separate sets to record the current search results: the current visited node set V and the candidate set C . Nodes in V are all visited nodes, while nodes in C have their distances computed from the query vector, but have not yet explored their neighbors.

Below, we give the details shown in Algorithm 4.

In the first phase, similar to that in the beam search on the proximity graph, we start the search at the entry point (line 1). At each step, we select a node with the smallest distance to q in C for expansion (line 3), considering only the edges labeled ID and $\neg$ ID (lines 4-5), akin to beam search. Notably, in this first phase, we eliminate the need for the beam width parameter w required in beam search. The search concludes when there are at least k nodes in V with distances to q that are smaller than all nodes in C .

Leveraging the results from the first phase, we can apply the method introduced in Section 5 to distinguish whether the query is an OOD or ID query by comparing the distance between the query vector and its current k -ANN, i.e., k -NN of vectors in V , with the precomputed value τ during the index construction (line 6).

Next, the search comes to the second phase. If the query is identified as OOD, we introduce another parameter ϕ , which will be discussed in the subsequent section. This parameter is utilized to regulate the cost by limiting the traversal of nodes in the proximity graph, thereby restricting the number of distance computations (line 16). Note that OOD queries cannot terminate earlier than ID queries; therefore, the termination condition for ID queries (line 12) is also a necessary condition for terminating OOD search (line 16). For OOD queries, the second phase traverses edges labeled ID and OOD on the augmented proximity graph (lines 13-15). If the query is identified as ID, we implement the same beam search strategy using the beam width w . The search terminates when there are at least w nodes in V whose distances to q are smaller than those of all nodes in C . In this case, the second phase traverses edges labeled ID and $\neg$ ID (lines 9-11), which together represent the original index built on X . Finally, when the search terminates, the top- k nodes in V with the smallest distances to q are returned (line 17).

As follows, we demonstrate that Algorithm 4 yields identical results for ID queries as Algorithm 1.

Theorem 6.1: *Given the same input q, k , and $w(\geq k)$, Algorithm 4 on the augmented graph G^+ returns the same result as Algorithm 1 on the proximity graph G if $\hat{\tau} \leq \tau$ holds at line 9 of Algorithm 4.*

Proof Sketch: If $\hat{\tau} \leq \tau$, the first while-loop stops when $\{v \in V \mid \forall u \in C, \delta(v, q) < \delta(u, q)\} \geq k$, and the second while-loop stops when this quantity reaches w . Since $w \geq k$, the two while-loops can be viewed as a single loop that terminates when $\{v \in V \mid \forall u \in C, \delta(v, q) < \delta(u, q)\} \geq w$. Now consider any node $u \in V$ whose distance to q can never be within the top- w . Then at least w nodes in V are closer to q than u . As a node inserted into V are also inserted into C , u will never be explored, as the condition $\{v \in V \mid \forall u \in C, \delta(v, q) < \delta(u, q)\} \geq w$ must be satisfied before expanding u . Thus, whenever $|V| > w$, the farthest nodes in V can be removed without affecting the result, so V can be regarded as a beam of size at most w . Under this view, C is unnecessary: nodes in V can be marked visited directly upon expansion, making V equivalent to the set S in Algorithm 1. For the termination condition of the while loop, as C comprises unvisited nodes in V and V has a maximum size of w , the condition $\{v \in V \mid \forall u \in C, \delta(v, q) < \delta(u, q)\} \geq w$ implies that all nodes in V are visited, mirroring line 2 in Algorithm 1. $\square$

Based on Theorem 6.1, if the top- k nodes with the smallest distances to q in V are directly returned at line 6 in Algorithm 4, the results are identical to those obtained when setting $w = k$ in Algorithm 1. Given that in the first phase, where the goal is to reach the region containing one of the k -NN of a query q , a large queue size is unnecessary, aligning well with the choice of a smaller beam width in the first phase of our search.

6.2 Iterative Batch Index Augmentation

In this section, we discuss how to set ϕ so that OOD queries can consistently obtain high-quality k -ANN results.

At a high level, OOD queries are grouped into batches for scheduling index augmentation. Let β denote the batch size, i.e., the number of OOD queries grouped into each index-augmentation batch. All OOD queries in the same batch are searched on the same snapshot of the augmented proximity graph. After their search

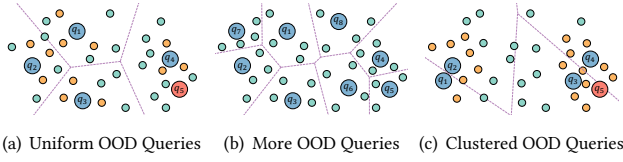


Figure 9: An Example of Batch Index Augmentation

results are obtained, we use them to augment the graph by Algorithm 3, and the updated graph is then used by the next batch. In this way, early OOD queries improve the graph for later ones, allowing ϕ to decrease over time. Compared with a sequential scheme that updates the graph after every OOD query, batching avoids blocking between consecutive queries. For example, as illustrated in Fig. 2, with $\beta = 2$, q_2 can be searched on the same graph snapshot as q_1 without waiting for the augmentation triggered by q_1 . Thus, ϕ is controlled batch-wise rather than query-wise, e.g., q_1 and q_2 have the same ϕ in Fig. 2, improving efficiency while preserving high-quality results.

We next discuss the details of ϕ . In our algorithm, ϕ is not an explicitly tuned parameter; rather, it is determined implicitly through the hyperparameter β , i.e., the batch size. For the first batch of β OOD queries, we set $\phi = n$ to acquire their k -ANN (effectively k -NN) by traversing all nodes in the proximity graph G . These results are then used to augment the proximity graph. After this first batch, the augmented graph can already guide a new OOD query toward the region of its nearest processed OOD query. For example, in Fig. 9(a), the small circles represent ID vectors in dataset X , and $q_1, \dots, q_4$ are processed OOD queries. The new edges introduced by these queries make their nearby regions easier to reach from the entry point. For a new OOD query such as q_5 , the search can first navigate to the region of its nearest processed OOD query, e.g., q_4 . However, connectivity within that region, especially near q_5 itself, may still be insufficient. It is because the region is divided by processed OOD queries; within the region, no node OOD queries are involved in augmenting the index. Thus, the second phase requires a ϕ to explore nodes inside that region for high-quality results.

Given that one beam search on the proximity graph typically takes empirically $O(d \ln n)$ time [9, 31, 35, 43], for simplicity in the subsequent analysis, we assume that beam search with any beam width on the proximity graph requires $O(d \ln n)$ time.

It remains an open problem how to model the distribution of OOD queries in general. Here, we adopt a uniform assumption only for analysis, following common theoretical treatments in prior vector search work [9, 35, 43]. Importantly, this case is conservative: when OOD queries are clustered, the augmentation needed is more localized, and many intermediate regions need not be traversed. This intuition is illustrated in Fig. 9(c), where processed OOD queries form two clusters near q_1, q_2 and q_3, q_4 . In this case, many vectors in the intermediate region, such as the green nodes, need not be traversed, since later OOD queries remain concentrated near one of the two clusters, e.g., q_5 . By contrast, under a uniform OOD distribution, augmentation is spread more broadly across the index, since every region may contain OOD queries and require traversal. Separately, the arrival order of OOD queries may affect our approach. For example, OOD queries may arrive in a skewed order, such as cluster by cluster, rather than in a random order.

Algorithm 5: BatchAugment (G, Q, k, w, β)

Input : PG G , query set $Q = \{q_1, \dots, q_m\}$, k for top- k , beam width w , and batch size β
Output : augmented PG G^+

```

1  $Bcap \leftarrow \beta$ ;  $BID \leftarrow 0$ ;  $\phi \leftarrow n$ ;
2  $G_0^+ \leftarrow G$ ;  $G_1^+ \leftarrow G$ ;  $M \leftarrow$  the out-degree limit of  $G$ ;
3 for each  $q \in Q$  do
4    $\mathcal{N}(q) \leftarrow \text{TwoPhaseSearch}(G_{BID}^+, q, k, w, \phi)$ ;
5   if  $\hat{\tau} > \tau$  in line 8 when calling Algorithm 4 then
6      $p_1(q) \leftarrow$  top-1 nearest node to  $q$  in  $V$  at line 7 in Algorithm 4;
7      $G_{BID+1}^+ \leftarrow \text{AugmentPG}(G_{BID+1}^+, q, p_1(q), \mathcal{N}(q), M)$ ;
8      $Bcap \leftarrow Bcap - 1$ ;
9     if  $Bcap \leq 0$  then
10       $Bcap \leftarrow \beta$ ;  $BID \leftarrow BID + 1$ ;
11       $\phi \leftarrow \min(n, \lceil \ln n + n/\beta/BID \rceil)$ ;  $G_{BID+1}^+ \leftarrow G_{BID}^+$ ;
11 return  $G_{BID+1}^+$ ;
```

Although our analysis assumes a uniform distribution, PFIndex remains effective under such skewed workloads by varying the β value. Empirical evaluation is provided in Section 7 (Exp. 4), where OOD queries arrive cluster by cluster in a skewed order.

Under uniform assumption, after β OOD queries have been processed, the dataset X can be viewed as divided into β regions, each containing about n/β nodes. To ensure high-quality k -ANN results for a new OOD query, we set ϕ to traverse all nodes in one such region in the second phase. Since the first phase is beam search with beam width k , which is assumed to traverse $O(\ln n)$ nodes, we set $\phi = \ln n + n/\beta$ for a new OOD query. Upon completing the processing of the second batch of queries, we can utilize k -ANN results to enhance the proximity graph further. For example, in Fig. 9(b), incorporating more processed OOD queries improves connectivity among the k -ANN sets of OOD queries and reduces the size of each region, i.e., containing $n/(2\beta)$ nodes, enabling a decrease in the ϕ value in subsequent batches. Specifically, after the $(i-1)$ -th batch, each region contains about $n/((i-1)\beta)$ nodes. Accordingly, for the i -th batch of β OOD queries, we set $\phi = \ln n + \frac{n}{(i-1)\beta}$. Note that for later incoming queries, ϕ may not be effective, as the termination condition at line 9 in Algorithm 4 could be more stringent than ϕ . This is expected: as more OOD queries are incorporated into the index, OOD search becomes increasingly similar to ID search.

Based on the above analysis, Algorithm 5 summarizes our iterative batch augmentation procedure. We initialize the current batch capacity $Bcap$ as β , the batch ID BID as 0, and set $\phi = n$ for the first batch of OOD queries (line 1). We also initialize the augmented proximity graph G_0^+ for batch 0 and G_1^+ for batch 1 as G with all edges labeled ID, and obtain the out-degree limit M from G (line 2). Then, for each query $q \in Q$, we invoke `TwoPhaseSearch` on G^+ with parameter ϕ (lines 3-4). If q is identified as an OOD query, i.e., $\hat{\tau} > \tau$ at line 8 of Algorithm 4, we obtain $p_1(q)$ from the search result, decrease $Bcap$, and use the returned k -ANN result to augment G_{BID+1}^+ for the next batch via Algorithm 3 (lines 5-10). When the current batch is exhausted, we move to the next batch by resetting $Bcap$, increasing BID , updating ϕ accordingly (line 10).

Note that the index augmentation process can be performed asynchronously with the search. Let $G^{(i-1)}$ denote the latest completed

Table 2: Statistics of Datasets

Dataset	Dim.	#Base	#Train	#Test		Vector type	
				ID	OOD	ID	OOD
WebVid [2]	512	2,505,000	2,505,000	10,000	10,000	Video	Text
Text-to-Image [1]	200	10,000,000	10,000,000	10,000	10,000	Image	Text
LAION [36]	512	10,004,480	10,004,480	10,000	10,000	Image	Text

augmented graph before the i -th batch. All queries in this batch are processed using $G^{(i-1)}$, while their search results are submitted to background workers to construct updates for subsequent batches. Thus, a query does not wait for the augmentation triggered by other queries in the same batch. The augmentation process copies only the adjacency lists affected by the newly selected edges. For each affected vertex, edge insertion, label adjustment, and pruning are performed on a private copy of its neighbor list. The completed list is then published as a new version for later query batches, while queries that have already started continue accessing the previous version. Thus, synchronization is needed only when publishing the completed list version, and this publication does not block queries that are already reading the previous snapshot.

Value Choice of β : Consider that beam search empirically takes $O(d \ln n)$ time [9, 31, 35, 43], while the second phase with parameter ϕ takes $O(d\phi)$ time since $\phi > \ln n$ and directly limits the number of traversed nodes. In the worst case, all m queries are OOD queries. Then the total search time is $O(d \cdot (\beta \cdot n + \beta \cdot (\ln n + n/\beta) + \dots + \beta \cdot (\ln n + n/((\lceil m/\beta \rceil - 1)\beta)))) = O(d \cdot (\beta \cdot n + m \ln n + n \log(\lceil m/\beta \rceil)))$. In contrast, in the best case where all queries are ID queries, the total search time is $O(d \cdot m \ln n)$. The same complexity also applies when we already have a proximity graph constructed for OOD queries [4, 14], since OOD search then becomes similar to ID search. To keep the worst-case complexity aligned with these optimal cases, we set $\beta = \ln n$, assuming $O(n) = O(m)$.

Time Complexity: Setting $\beta = \ln n$ and assuming $O(n) = O(m)$, the total search time becomes $O(d \cdot (\beta \cdot n + m \ln n + n \log(m/\beta))) = O(d \cdot n \ln n)$. For the augmentations, the cost arises from Algorithm 2, where the time required for lines 2-10 in Algorithm 3 is $O(|N_q|)$. Since N_q is obtained from the search, this cost is smaller than $O(\ln n)$ empirically. Algorithm 2 requires up to $O(|N_q|^2 d) \leq O(d \cdot \ln n^2)$ time. Hence, the total time needed is $O(d \cdot n \ln n^2)$. Under a multi-core execution model with $\Theta(\ln n)$ background workers, the augmentation work can be parallelized to an $O(d \cdot n \ln n)$ critical-path cost and largely overlapped with query processing. Therefore, the parallel wall-clock cost of answering the queries and augmenting the index remains approximately $O(d \cdot n \ln n)$, matching the asymptotic cost of query processing alone in the optimal case.

7 Experiments

Datasets: We utilize three large-scale cross-modal datasets with associated queries, following prior OOD vector search work [4, 14, 18]. The datasets are summarized in Table 2, where Dim. denotes the vector dimensionality, #Base is the number of base vectors, #Train is the number of OOD queries used for index construction, and #Test (ID/OOD) is the number of evaluation queries. We deduplicate the OOD queries so that the test OOD queries are disjoint from the train OOD queries, and the test ID queries are also disjoint from the base ID vectors. In Text-to-Image dataset, the base data and ID queries are image vectors generated by SE-ResNeXt-101 [13], while OOD queries are text vectors generated by a variant of DSSM [15];

the two modalities are mapped into a shared space via contrastive learning. In LAION dataset, the base data and ID queries are image embeddings, while the OOD queries are text embeddings. All embeddings are generated by CLIP-ViT-B/32 [36]. In WebVid dataset, the base data vectors are video embeddings and the query vectors are text embeddings, both generated by CLIP-ViT-B/32 [36]. Following prior OOD vector search work [4, 14, 18], we use the same released datasets, where the vectors are already normalized. Thus, we use inner product as distance metric in all experiments.

Algorithms: We compare PFIIndex with three state-of-the-art approaches, including HNSW, one of the state-of-the-art approaches for ID query vectors, and RoarGraph and NGFix, two existing approaches proposed for OOD query vectors as discussed in Section 3. We exclude the comparison with RobustVamana, as prior research has already demonstrated its inferior performance [4]. For each method, we employ a grid search (set recommended parameters if provided as starting points) to tune the parameters on each dataset for a Pareto-optimal recall-QPS curve. (1) NGFix [14], we set $M_{EX} = 48$ to limit the out-degree of each node. To construct its index, we first build an HNSW with $M = 32$ and $efC = 1000$, and following [14], only the bottom layer of HNSW is used as the base graph. Then, we use OOD queries to construct the new index. Following [14], we call NGFix twice: first with $N_q = 100$, $K_h = 100$, and $MaxS \leq 500$, and then with $N_q = 10$, $K_h = 10$, and $MaxS \leq 50$. For each OOD query, we set $N_q = 10$ to call RFix. The k -NN of each query used in index construction is computed by brute force using an A100 GPU. (2) RoarGraph [4], following [4, 14], we set $N_q = 100$ to control bipartite graph connections. For neighbor-aware projection and connectivity enhancement, we set $M = 35$ and $L = 2000$ to ensure high-quality neighbors. For fair comparison, we use an equivalent number of the entire OOD query vector set for index construction. The k -NN of each query used in index construction is computed by brute force using an A100 GPU. (3) HNSW [31], we set $M = 32$ and $efC = 1000$ to build a high-quality proximity graph for comparison. (4) PFIIndex, our approach. We set $\beta = 32$, $k = 100$, and $w = 1000$ by default. The batch mechanism is used only for scheduling index augmentation: all OOD queries in the same batch are searched on the same index snapshot, and the augmentation produced by that batch is applied only to subsequent batches. Each query is searched by a single thread, while background worker threads are used for index augmentation. To ensure fair comparison, the underlying proximity graph of PFIIndex uses the same HNSW parameters as those used in NGFix, i.e., $M = 32$ and $efC = 1000$.

Performance Metrics: Following existing works [4, 14, 18], we employ QPS (queries per second) and NDC (number of distance calculations) to measure efficiency, and use Recall and Rderr (relative distance error) to measure effectiveness. QPS is the number of queries processed per second, and NDC is the total number of distance calculations during the search. Recall is precisely defined in Section 2, while relative distance error (Rderr) is defined as $rderr@k = \text{avg}_{i=1}^k (\delta(\hat{p}_i, q) / \delta(p_i, q) - 1)$, where p_i is the i -NN of q and $\hat{p}_i$ is the i -ANN of q returned by algorithms. 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.

Experimental Environments: The experiments are conducted

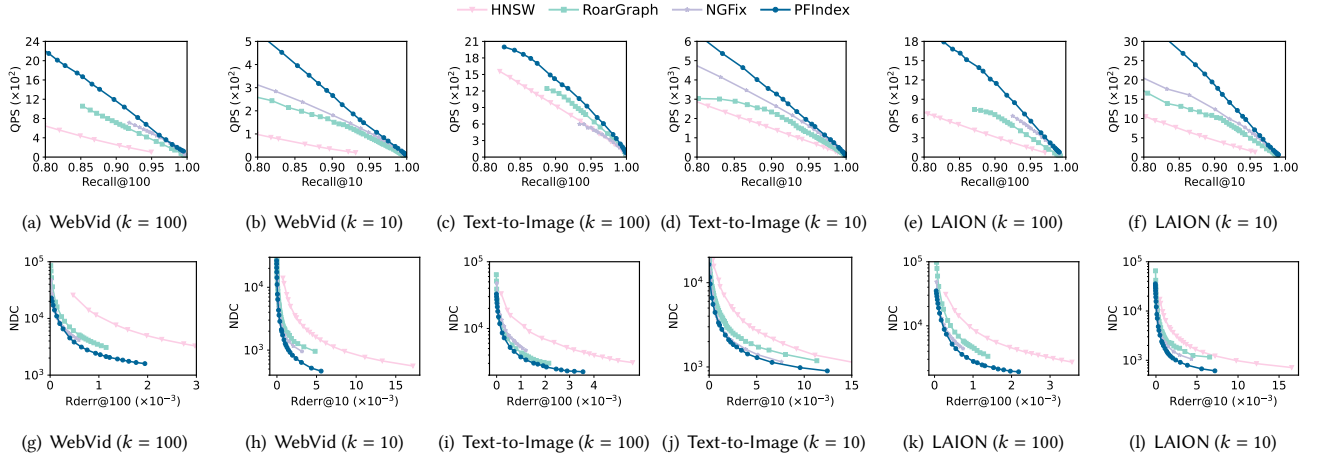


Figure 10: Search Performance on OOD Queries (Exp. 1)

on a CentOS Linux server with an Intel(R) Xeon(R) Silver 4215 CPU @ 2.50GHz and 128GB memory. All algorithms are implemented in C++ and compiled with g++ 11.4.1 using `-std=c++20`, `-Ofast`, and `-march=native`. Following NGFix implementation [14] and other OOD vector search implementations [4, 18], we use RaBitQ’s quantization functionality (with `-b_bits 8`) and SIMD intrinsics, including AVX512-related instructions as an implementation-level optimization to accelerate distance computation. For fairness, these system-level optimizations are applied to all compared methods.

For index construction of HNSW, RoarGraph and NGFix, we report the index time/size on the same single machine by utilizing multiple threads. For PFIndex, we perform searches for each query in a single thread. While for augmenting proximity graphs, each query is searched by a single thread, while background index augmentation is executed asynchronously using the remaining available threads. For vector search on ID and OOD test queries, we set $k = 10$ and $k = 100$ to evaluate the performance of the search algorithms using a single thread following the previous works [4, 14, 18], and incrementally adjust parameters within the search algorithm to achieve the desired recall.

Exp. 1: Search Performance on OOD Queries. In this experiment, all methods use all available threads to construct/augment their indexes using the train OOD queries, and their search performance is evaluated on the disjoint test queries using a single thread. As shown in Fig. 10, PFIndex consistently outperforms all baselines on OOD queries. Specifically, at $recall@100 = 0.95$, QPS of PFIndex is 1.36 – 1.48x that of NGFix, 1.25 – 1.77x that of RoarGraph, and 1.64 – 4.75x that of HNSW. HNSW, a state-of-the-art approach for ID queries, exhibits poor support for OOD queries, resulting in the lowest performance. NGFix usually outperforms RoarGraph, but is weaker on Text-to-Image when $k = 100$. As shown in Fig. 10(i), on Text-to-Image with $k = 100$, NGFix achieves much better result quality than HNSW, but only slightly higher recall, suggesting that its add-edge strategy may fail to introduce sufficiently high-quality edges in some OOD scenarios. PFIndex consistently performs the best, mainly due to its two-phase search algorithm for OOD queries, which provides finer control through ϕ . Its advantage is especially clear at moderate recall, around 0.9, where most relevant neighbors can be identified quickly. When recall approaches 1, all methods

Table 3: Index Cost of Different Approaches (Exp. 2)

Algo.	HNSW	RoarGraph	NGFix	PFIndex
Dataset	Index build time (sec)			
WebVid	909	(666+),3,046	(666+),3,984	(909+),1,228
Text-to-Image	2,745	(10,599+),10,710	(10,599+),7,178	(2,745+),4,372
LAION	4,299	(14,288+),16,501	(14,288+),11,792	(4,299+),5,391
Dataset	Total index size for joint ID/OOD support (MB)			
WebVid	5,174	(5,174+),5,307	(5,174+),5,404	(5,174+),1,523
Text-to-Image	9,109	(9,109+),9,167	(9,109+),9,116	(9,109+),4,933
LAION	20,761	(20,761+),20,589	(20,761+),21,976	(20,761+),7,072
Dataset	Index size for OOD-only support (MB)			
WebVid	5,174	5,307	5,404	5,219
Text-to-Image	9,109	9,167	9,116	9,113
LAION	20,761	20,589	21,976	20,814

become closer, because achieving very high recall requires large search parameters and traversal of many nodes in the graph.

Exp. 2: Index Cost of Different Approaches. Following the same experimental settings in Exp.1, Table 3 reports the index build time and index size of different approaches.

For build time, HNSW is the fastest because its construction is the simplest: it only needs to obtain neighbors for each vector and then apply RNG pruning. For RoarGraph and NGFix, the parenthesized numbers denote the time for computing the ground-truth k -NN of the train OOD queries, using an A100 GPU; the cost would be much higher on CPU alone. Even excluding this GPU-assisted k -NN computation, both RoarGraph and NGFix still require substantially more time than HNSW and PFIndex. It is important to note that the reported time for PFIndex includes both the time taken to respond to train OOD queries and index augmentation time, where the times in parentheses represent the HNSW build time. By achieving a comparable time scale to HNSW, our approach not only delivers high-quality responses to trained OOD queries (Exp. 3) but also constructs an augmented proximity graph capable of supporting both ID and OOD queries (Exps. 1, 4, and 5). We further increase the HNSW construction budget by setting $efC = 3000$ while keeping $M = 32$. Its construction time increases to 2,367, 7,390, and 9,734 seconds on WebVid, Text-to-Image, and LAION datasets, respectively, larger than the total build time of PFIndex. Nevertheless, at $recall@10 = 0.95$, its QPS improves only from 249.4/804.9/164.1 to

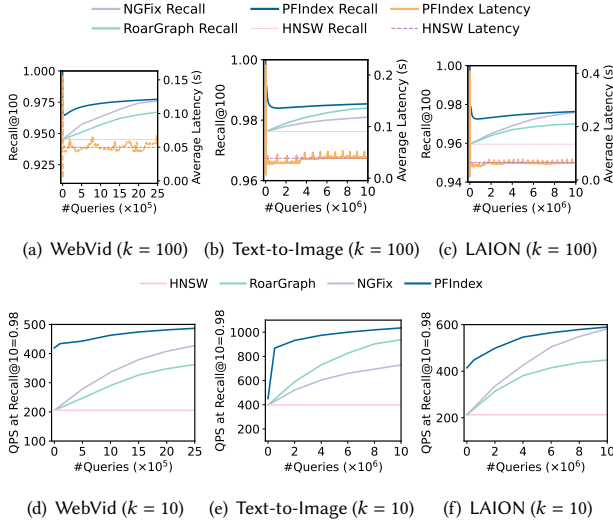


Figure 11: Performance of Online OOD Queries (Exp. 3)

250.0/805.2/164.3, showing that additional construction effort alone does not resolve the OOD search issues.

For index size, Table 3 reports two deployment settings. In the joint ID/OOD setting, RoarGraph and NGFix require an additional HNSW index to retain competitive performance on ID queries, since their OOD-oriented indexes degrade ID-query performance, as shown in Exp. 6. Their total sizes are therefore the sums of the HNSW index and their respective OOD-oriented indexes. PFIIndex also operates on top of HNSW, and its reported total size includes this base graph. However, PFIIndex reuses the same graph for both query types and stores only the differential edges labeled $\neg$ ID and OOD. As a result, PFIIndex reduces the total memory by up to 36.7% compared with the two-index deployments. For OOD-only search, PFIIndex does not need to preserve the separate ID-oriented graph view. We preserve the graph containing the edges labeled ID or OOD and discard the $\neg$ ID edges. The resulting index sizes are comparable to RoarGraph and NGFix indexes. Therefore, the primary storage advantage of PFIIndex arises when both ID and OOD queries must be efficiently supported.

Exp. 3: Performance when Online Processing of OOD Queries.

We evaluate how search performance evolves as OOD queries continuously arrive by using the train dataset, as shown in Fig. 11, where the horizontal axis reports the cumulative number of OOD queries observed before evaluation. Since RoarGraph and NGFix are prior-dependent methods, we adapt them to the online setting. At each checkpoint after the first t OOD queries have arrived, we use only these t queries as their prior query set and rebuild the corresponding index. The rebuilt index is then evaluated on the next unseen queries. For Fig. 11(a)–(c), we set the beam width as 1000 for all methods, and set $\beta = 32$ for PFIIndex, while in Fig. 11(d)–(f), we set the minimal beam width required for different methods to achieve $recall@10 = 0.98$. PFIIndex initially uses a relatively large dynamically determined ϕ to obtain high-quality results for early queries. Thus, its initial query latency is higher (or initial QPS is lower), but remains below 0.15, 0.2, and 0.4 seconds on WebVid, Text-to-Image, and LAION, respectively. As more OOD queries are processed, the augmented graph becomes increasingly effective,

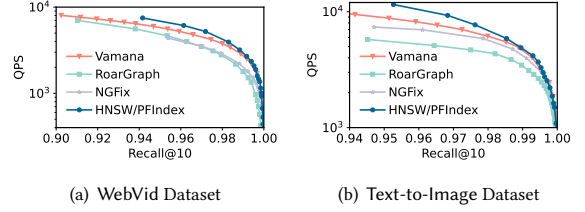


Figure 12: Search Performance on ID Queries (Exp. 4)

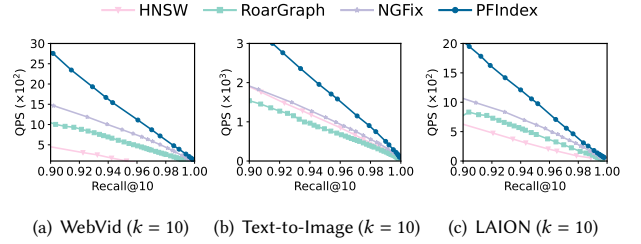


Figure 13: Performance on Mixed ID/OOD Queries (Exp. 5)

allowing ϕ to decrease and improving query efficiency while maintaining high recall. Although the search performance of RoarGraph and NGFix also improves when more arrived queries are used as prior information, their prefix-trained indexes remain less effective when the query volume is insufficient. This result demonstrates that simply rebuilding a prior-dependent index from the limited queries observed so far cannot provide the same online effectiveness as the progressive augmentation of PFIIndex. Moreover, their latency is much higher than PFIIndex, since their online processing includes the index rebuilding cost; the repeated construction cost needs to be amortized over the newly served queries.

Exp. 4: Search Performance on ID Queries. We use the base dataset to build indices and evaluate the search performance on disjoint test ID queries. We additionally include Vamana [19] as an ID-oriented baseline. Fig. 12 shows the results on WebVid and Text-to-Image, while the results on LAION are reported in Fig. 6. PFIIndex achieves the same search performance as HNSW because it preserves the full HNSW graph structure for ID search, i.e., the edges labeled ID and $\neg$ ID in the augmented graph. Therefore, like HNSW, PFIIndex consistently performs the best on ID queries. Both RoarGraph and NGFix perform worse on ID queries, with RoarGraph slightly weaker than NGFix. This difference mainly arises because NGFix modifies the bottom-layer HNSW graph and thus still benefits from the original ID-oriented structure, whereas RoarGraph is constructed mainly for OOD support and only later improves connectivity using dataset vectors.

Exp. 5: Search Performance on Mixed ID/OOD Queries. We use the train dataset to build indices and evaluate mixed ID/OOD workloads using multiple random shuffles with different seeds on the test dataset and report the average results. As shown in Fig. 13, PFIIndex accurately distinguishes ID and OOD queries with 100% precision on all datasets, leading to stable performance close to that on pure OOD workloads while also benefiting from the relative ease of ID queries. HNSW performs better than in the pure-OOD setting because it is strong on ID queries. In contrast, RoarGraph and NGFix do not distinguish between ID and OOD queries, but

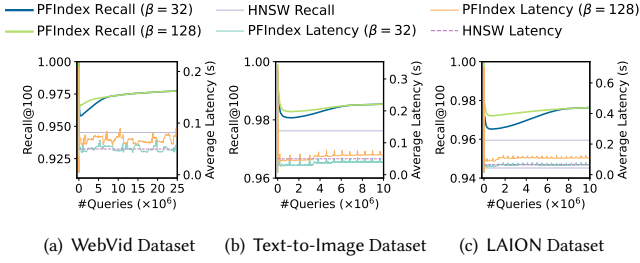


Figure 14: Performance on Skewed Distribution (Exp. 6)

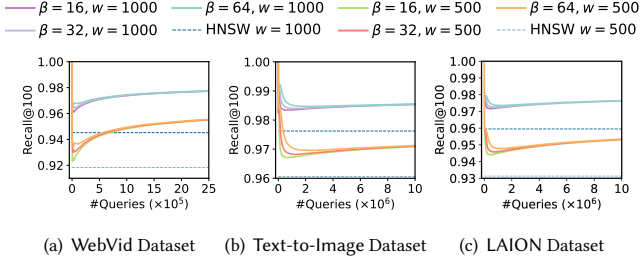


Figure 15: Effect of Different Parameters (Exp. 7)

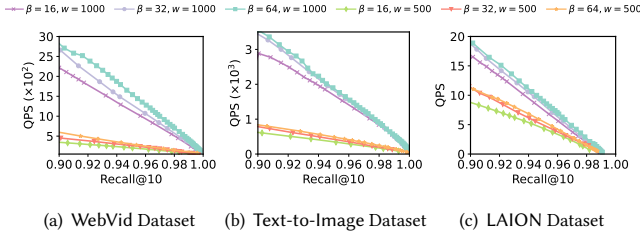


Figure 16: Effect of Different Parameters on Index (Exp. 7)

handle all queries using the same index and search strategy, which leads to inferior performance under mixed workloads.

Exp. 6: Performance under Skewed Online Processing of OOD Queries. In this experiment, we further evaluate a less favorable online setting by using the train dataset, where OOD queries are first clustered, and queries from the same cluster arrive consecutively before moving to the next cluster, thereby forming a skewed arrival order. Compared with the random order in Exp. 3, this setting is less friendly to PFIndex, since index augmentation is driven by recently processed queries, while the query distribution may shift significantly when the stream moves from one cluster to another. The results on three datasets are shown in Fig. 14. PFIndex remains effective under this skewed ordering, and the hyperparameter β plays a key role in handling this harder case. Specifically, a larger β allows more OOD queries to scan more vectors in the early stage, which improves the cumulative recall during online processing and provides stronger augmentation for subsequent queries after the stream shifts to new clusters. This benefit comes at the cost of slightly higher processing overhead, and thus higher latency.

Exp. 7: Effect of Different Parameters. We vary two parameters, β and w in index augmentation (Algorithm 5), to evaluate their effects on search performance. In Fig. 15, we use the train dataset as an online stream for evaluation. In Fig. 16, we first construct the index using the train dataset and then evaluate it on the disjoint test queries. Fig. 15 shows their effect on online OOD processing for

Table 4: Sensitivity of ID/OOD Distinction (Exp. 8)

Dataset	τ value	$0.9\tau_0$	$0.95\tau_0$	τ_0	$1.05\tau_0$	$1.1\tau_0$
WebVid	false-ID	1.26%	0.16%	0%	0%	0%
	false-OOD	0%	0%	0%	0%	0%
Text-to-Image	false-ID	0.42%	0.08%	0%	0%	0%
	false-OOD	0%	0%	0%	0%	0%
LAION	false-ID	1.85%	0.23%	0%	0%	0%
	false-OOD	0%	0%	0%	0%	0.02%

the train OOD queries, where recall is reported cumulatively over all processed OOD queries. A larger β causes more node traversal for early queries and thus yields slightly higher recall, while w has a much larger effect because it directly controls the quality of the returned k -ANN results during online processing. When w is fixed, the recall curves under different β values gradually become similar as more OOD queries are processed, because the augmented graph becomes sufficiently strong and the effect of batch size diminishes. Moreover, PFIndex consistently achieves higher recall than directly using HNSW to serve OOD queries. Fig. 16 shows the effect of β and w on the quality of the augmented graph, evaluated using test OOD queries. The results indicate that w is the dominant factor, since it directly determines the quality of the online k -ANN results used for augmentation. Higher-quality k -ANN results from the train OOD queries lead to a stronger augmented graph and thus better performance on the test OOD queries. In contrast, β affects only a small fraction of the train OOD queries, mainly the early ones, and therefore has only a minor impact on the final index quality.

Exp. 8: Sensitivity to the Threshold τ . We vary the threshold τ in $\{0.9\tau_0, 0.95\tau_0, \tau_0, 1.05\tau_0, 1.1\tau_0\}$ to show its effect on distinguishing ID and OOD queries, where τ_0 is computed by the base dataset, and evaluation is on the disjoint test dataset. Table 4 reports the false-ID and false-OOD rates on three datasets. When $\tau = \tau_0$, both false-ID and false-OOD rates are zero on all datasets, indicating that τ_0 is an effective default choice. When τ is smaller than τ_0 , the false-ID rate increases slightly because a stricter threshold causes a few ID queries to be classified as OOD, while the false-OOD rate remains zero. When τ is larger than τ_0 , the false-ID rate remains zero, and the false-OOD rate is also zero except for a negligible value of 0.02% on LAION at $1.1\tau_0$.

8 Conclusion

In this paper, we present a prior-free index for cross-modal vector search on both ID and OOD queries. Our extensive experiments demonstrate the superiority of our approach in terms of accuracy and efficiency. We achieve the best search performance for OOD queries while maintaining comparable performance for ID queries compared to state-of-the-art methods for OOD and ID, respectively. For index cost, PFIndex achieves up to 3.18x faster index construction and up to 36.7% lower total index memory when jointly supporting ID and OOD queries, while maintaining an OOD-only index size comparable to those of the existing OOD-oriented approaches.

Important directions for future work include automatically detecting workload skewness or distribution drift and adaptively tuning β to adjust ϕ , as well as extending PFIndex to multi-vector late-interaction retrieval and investigating whether index-relative OOD persists in modern models such as ColQwen [7].

References

- [1] Artem Babenko and 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>.
- [2] 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 *ICCV*. IEEE, 1708–1718.
- [3] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011*. University of Miami, 591–596.
- [4] Meng Chen, Kai Zhang, Zhenying He, Yanan Jing, and X. Sean Wang. 2024. RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 11 (2024), 2735–2749.
- [5] Yannis Chronis, Helena Caminal, Yannis Papakonstantinou, Fatma Özcan, and Anastasia Ailamaki. 2025. Filtered vector search: State-of-the-art and research opportunities. *Proceedings of the VLDB Endowment* 18, 12 (2025), 5488–5492.
- [6] Wei Dong, Moses Charikar, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011*. ACM, 577–586.
- [7] Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. 2025. ColPali: Efficient Document Retrieval with Vision Language Models. In *ICLR*. OpenReview.net.
- [8] Cong Fu, Changxu Wang, and Deng Cai. 2022. High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 8 (2022), 4139–4150.
- [9] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474.
- [10] Yujian Fu, Cheng Chen, Yao Chen, Weng-Fai Wong, and Bingsheng He. 2025. Vista: Vector Indexing and Search for Large-Scale Imbalanced Datasets. In *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 543–556.
- [11] 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 (2024), 167.
- [12] Ben Harwood and Tom Drummond. 2016. FANNG: Fast Approximate Nearest Neighbour Graphs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*. IEEE Computer Society, 5713–5722.
- [13] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-Excitation Networks. In *CVPR*. Computer Vision Foundation / IEEE Computer Society, 7132–7141.
- [14] 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 (2025), 1–28.
- [15] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*. ACM, 2333–2338.
- [16] Siteng Huang, Biao Gong, Yulin Pan, Jianwen Jiang, Yiliang Lv, Yuyuan Li, and Donglin Wang. 2023. VoP: Text-Video Co-Operative Prompt Tuning for Cross-Modal Retrieval. In *CVPR*. IEEE, 6565–6574.
- [17] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A Survey on Locality Sensitive Hashing Algorithms and their Applications. *CoRR abs/2102.08942* (2021).
- [18] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankut Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. 2022. OOD-DiskANN: Efficient and Scalable Graph ANNS for Out-of-Distribution Queries. *CoRR abs/2211.12850* (2022).
- [19] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc.
- [20] Leonid V. Kantorovich. 1960. Mathematical Methods of Organizing and Planning Production. *Management Science* 6 (1960), 366–422.
- [21] Jie Lei, Linjie Li, Luowei Zhou, Zhe Gan, Tamara L. Berg, Mohit Bansal, and Jingjing Liu. 2021. Less Is More: ClipBERT for Video-and-Language Learning via Sparse Sampling. In *CVPR*. Computer Vision Foundation / IEEE, 7331–7341.
- [22] Patrick S. H. 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 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.
- [23] Binhong Li, Xiao Yan, and Shangqi Lu. 2025. Fast-Convergent Proximity Graphs for Approximate Nearest Neighbor Search. *CoRR abs/2510.05975* (2025).
- [24] Mocheng Li, Xiao Yan, Baotong Lu, Yue Zhang, James Cheng, and Chenhao Ma. 2025. Attribute Filtering in Approximate Nearest Neighbor Search: An In-depth Experimental Study. *Proc. ACM Manag. Data* 3, 6 (2025), 1–26.
- [25] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Trans. Knowl. Data Eng.* 32, 8 (2020), 1475–1488.
- [26] 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 *NeurIPS*.
- [27] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. 2024. RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval. *CoRR abs/2409.10516* (2024).
- [28] Song Liu, Haoqi Fan, Shengsheng Qian, Yiru Chen, Wenkui Ding, and Zhongyuan Wang. 2021. HiT: Hierarchical Transformer with Momentum Contrast for Video-Text Retrieval. In *ICCV*. IEEE, 11895–11905.
- [29] Yaowen Liu, Xuejia Chen, Anxin Tian, Haoyang Li, Qinbin Li, Xin Zhang, Alexander Zhou, Chen Jason Zhang, Qing Li, and Lei Chen. 2026. GPU-Accelerated Algorithms for Graph Vector Search: Taxonomy, Empirical Study, and Research Directions. *CoRR abs/2602.16719* (2026).
- [30] P. C. Mahalanobis. 1918. On the Generalised Distance in Statistics. *Sankhya A* 80, 1 (12 1918), 1–7. Reprint of the 1936 original.
- [31] Yuri A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.
- [32] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *27th Annual Conference on Neural Information Processing Systems 2013*. 3111–3119.
- [33] Javier Alvaro Vargas Muñoz, Marcos André Gonçalves, Zanoni Dias, and Ricardo da Silva Torres. 2019. Hierarchical Clustering-Based Graphs for Large Scale Approximate Nearest Neighbor Search. *Pattern Recognit.* 96 (2019).
- [34] Nasser M Nasrabadi and Robert A King. 1988. Image coding using vector quantization: A review. *IEEE Transactions on communications* 36, 8 (1988), 957–971.
- [35] 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 (2023), 54:1–54:27.
- [36] 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 *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 8748–8763.
- [37] Simon Schrod, David T. Hoffmann, Max Argus, Volker Fischer, and Thomas Brox. 2025. Two Effects, One Trigger: On the Modality Gap, Object Bias, and Information Imbalance in Contrastive Vision-Language Models. In *ICLR*. OpenReview.net.
- [38] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xianguo Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *SIGMOD '21*. ACM, 2614–2627.
- [39] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 14, 11 (2021), 1964–1978.
- [40] 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. *Data Engineering* (2023), 3–21.
- [41] Jiadong Xie, Yingfan Liu, and Jeffrey Xu Yu. 2026. Advances of Query Processing in Vector Databases. *Proceedings of the VLDB Endowment* 19, 12 (2026), 4893–4899.
- [42] Jiadong Xie, Jeffrey Xu Yu, and Yingfan Liu. 2025. Fast Approximate Similarity Join in Vector Databases. *Proc. ACM Manag. Data* 3, 3 (2025), 158:1–158:26.
- [43] Jiadong Xie, Jeffrey Xu Yu, and Yingfan Liu. 2025. Graph Based K-Nearest Neighbor Search Revisited. *ACM TODS* (May 2025).
- [44] Shuo Yang, Jiadong Xie, Yingfan Liu, Jeffrey Xu Yu, Xiyue Gao, Qianru Wang, Yanguo Peng, and Jiangtao Cui. 2025. Revisiting the Index Construction of Proximity Graph-Based Approximate Nearest Neighbor Search. *PVLDB* 18, 6 (2025), 1825–1838.
- [45] Ziqi Yin, Jianyang Gao, Pasquale Balsebre, Gao Cong, and Cheng Long. 2025. DEG: Efficient Hybrid Vector Search Using the Dynamic Edge Navigation Graph. *Proc. ACM Manag. Data* 3, 1 (2025), 29:1–29:28.
- [46] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. 2023. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023*. USENIX Association, 377–395.
- [47] Hongbin Zhong, Matthew Lentz, Nina Narodytska, Adriana Szekeres, and Kexin Rong. 2025. HoneyBee: Efficient Role-based Access Control for Vector Databases via Dynamic Partitioning. *CoRR abs/2505.01538* (2025).
- [48] Shurui Zhong, Dingheng Mo, and Siqiang Luo. 2025. LSM-VEC: A Large-Scale Disk-Based System for Dynamic Vector Search. *CoRR abs/2505.17152* (2025).