

FGIM: a Fast Graph-based Indexes Merging Framework for Approximate Nearest Neighbor Search

ZEKAI WU, East China Normal University, China

JIABAO JIN, Ant Group, China

PENG CHENG*, Tongji University, China

XIAOYAO ZHONG, Ant Group, China

LEI CHEN, HKUST (GZ), China and HKUST, China

YONGXIN TONG, Beihang University, China

ZHITAO SHEN, Ant Group, China

JINGKUAN SONG, Tongji University, China

HENG TAO SHEN, Tongji University, China

XUEMIN LIN, Shanghai Jiaotong University, China

As the state-of-the-art methods for high-dimensional data retrieval, Approximate Nearest Neighbor Search (ANNS) approaches with graph-based indexes have attracted increasing attention and play a crucial role in many real-world applications, e.g., *retrieval-augmented generation* (RAG) and recommendation systems. Unlike the extensive works focused on designing efficient graph-based ANNS methods, this paper delves into merging multiple existing graph-based indexes into a single one, which is also crucial in many real-world scenarios (e.g., cluster consolidation in distributed systems and read-write contention in real-time vector databases). We propose a Fast Graph-based Indexes Merging (FGIM) framework with three core techniques: (1) *Proximity Graphs* (PGs) to *k* *Nearest Neighbor Graph* (*k*-NNG) transformation used to extract potential candidate neighbors from input graph-based indexes through *cross-querying*, (2) *k*-NNG refinement designed to identify overlooked high-quality neighbors and maintain graph connectivity, and (3) *k*-NNG to PG transformation aimed at improving graph navigability and enhancing search performance. Then, we integrate our FGIM framework with the state-of-the-art ANNS method, HNSW, and other existing mainstream graph-based methods to demonstrate its generality and merging efficiency. Extensive experiments on six real-world datasets show that our FGIM framework is applicable to various mainstream graph-based ANNS methods, achieves up to 3.5× speedup over HNSW’s incremental construction and an average of 7.9× speedup for methods without incremental support, while maintaining comparable or superior search performance.

CCS Concepts: • **Information systems** → **Storage architectures**.

Additional Key Words and Phrases: Graph-based Index, ANNS, Algorithms

*Corresponding Author.

Authors’ Contact Information: Zekai Wu, East China Normal University, Shanghai, China, zekaiwu@stu.ecnu.edu.cn; Jiabao Jin, Ant Group, Shanghai, China, jinjiabao.jjb@antgroup.com; Peng Cheng, Tongji University, Shanghai, China, cspcheng@tongji.edu.cn; Xiaoyao Zhong, Ant Group, Shanghai, China, zhongxiaoyao.zxy@antgroup.com; Lei Chen, HKUST (GZ), Guangzhou, China and HKUST, Hong Kong SAR, China, leichen@cse.ust.hk; Yongxin Tong, Beihang University, Beijing, China, yxtong@buaa.edu.cn; Zhitao Shen, Ant Group, Shanghai, China, zhitao.szt@antgroup.com; Jingkuan Song, Tongji University, Shanghai, China, jingkuan.song@gmail.com; Heng Tao Shen, Tongji University, Shanghai, China, shenhengtao@hotmail.com; Xuemin Lin, Shanghai Jiaotong University, Shanghai, China, xuemin.lin@gmail.com.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2836-6573/2026/2-ART37

<https://doi.org/10.1145/3786651>

ACM Reference Format:

Zekai Wu, Jiabao Jin, Peng Cheng, Xiaoyao Zhong, Lei Chen, Yongxin Tong, Zhitao Shen, Jingkuan Song, Heng Tao Shen, and Xuemin Lin. 2026. FGIM: a Fast Graph-based Indexes Merging Framework for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 37 (February 2026), 27 pages. <https://doi.org/10.1145/3786651>

1 Introduction

Approximate Nearest Neighbor Search (ANNS) [4, 5] is a fundamental problem across various fields such as machine learning [16, 55], information retrieval [53, 71], recommendation systems [17, 38], vector databases [54], and Large Language Models (LLMs) [6, 18]. Recent deep-learning methods have revolutionized data representation by embedding unstructured data (e.g., images and texts) into high-dimensional vectors that preserve semantic relationships, which are crucial for similarity-based retrieval and various downstream machine-learning tasks. ANNS can efficiently retrieve semantically related data points from large-scale vectorized datasets, and is indispensable for modern AI applications [6, 30, 35]. Given a vector dataset $\mathcal{X}$ and a query vector $\vec{x}_q \in \mathbb{R}^d$, ANNS aims to efficiently and effectively retrieve a vector $\vec{x}_r$ from $\mathcal{X}$ with the minimum distance to $\vec{x}_q$. According to recent studies [7, 33, 40, 56], graph-based indexes have emerged as one of the most effective techniques for ANNS in view of their superior search accuracy.

Despite the good performance of graph-based approaches, their practical deployment in industrial settings remains challenging. Industrial applications often involve massive datasets (e.g., 1B–10B vectors), where *distance computation* across candidate vectors dominates query latency. For small indexes constructed from data shards, merging them into a larger index can improve computational efficiency (e.g., Hierarchical Navigable Small World (HNSW) [37] consumes $O(\log n)$ time for search [56], and since $\sum \log n_i < \log \sum n_i$, a larger index enables faster search than partitioned ones). Moreover, real-world systems often operate in dynamic environments, where both data and device conditions change frequently. As a result, merging multiple indexes into a single one is required in various scenarios, which poses significant challenges.

We first consider a cost-effective and high-performance scenario of real-time industrial applications (e.g., *Retrieval-Augmented Generation* (RAG) [6, 30]), where indexing must be performed online to support continuous data writes. While HNSW [37] is widely adopted as a real-time indexing solution for its incremental insertion capability, its memory-intensive multi-layer structure motivates complementary SSD-based solutions (e.g., DiskANN [27]), which trade query speed for cost efficiency. An industrial hybrid approach, adopted by Milvus [54] and VSAG [70] illustrated in Figure 1, follows a Log-Structured Merge (LSM) architecture [44]: a size-bounded HNSW index (typically 1M–5M vectors) handles real-time writes, with periodic offloading to disk-based indexes. However, accumulating disk-based indexes requires background reconstruction to preserve search quality. *This process is computationally expensive, thus difficult to be applied in online scenarios.*

In another distributed scenario illustrated in Figure 2, to handle large-scale data efficiently, distributed storage and indexing systems [31, 64] have become increasingly prevalent. In these systems, each node constructs its index based on locally available data. As the distributed system evolves, scaling requirements may arise due to *cluster consolidation* [24, 34] or *resource reallocation* [29]. Specifically, computing nodes within a cluster are often merged to reduce costs and simplify management, which requires merging the existing indexes into a single one. The existing methods [19, 22, 45] rely on dataset-oriented merging operations (i.e., rebuilding the index from scratch) due to their unique construction strategies, which incurs *substantial computational overhead and limited scalability*.

Challenges. To the best of our knowledge, no existing studies [19, 20, 22, 27, 37, 45] can meet the aforementioned requirements of industrial deployment to efficiently merge the graph-based indexes.

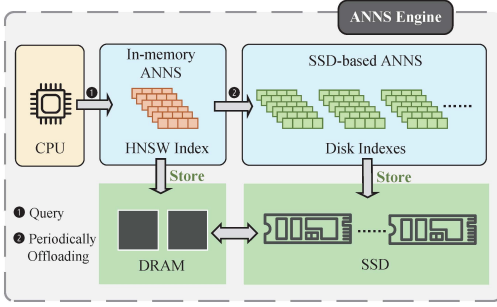


Fig. 1. A hybrid engine for ANNS.

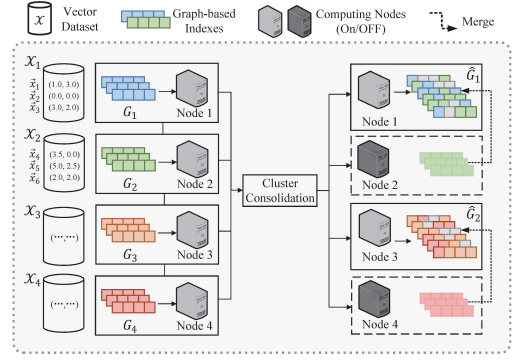


Fig. 2. An illustration of cluster consolidation.

To bridge this gap, we develop a practical approach to significantly reduce the time overhead of the merging process assuming the involved graph-based indexes fit in main memory, which is realistic in modern systems. While vectors may reside on SSDs, merging is typically performed on bounded, memory-resident graph shards [70], and growing DRAM capacity makes such in-memory consolidation increasingly feasible [8, 56]. Our work focuses on the efficient merging of multiple existing graph indexes into a single one, which is non-trivial and has two key challenges.

Challenge I: *How can the neighbor relationships stored in the existing indexes be used to facilitate the construction of the merged graph-based index?* Intuitively, an effective merging approach should extract valuable information (e.g., neighborhood relationship) from the existing multiple indexes to avoid as many computational overheads as possible, which is particularly important for high-dimensional datasets (e.g., GloVe [46], Gist [28] and Crawl [3]). However, different ANNS indexing structures employ different graph construction strategies and pruning techniques [56]. To ensure that, our approach is compatible with various indexing structures, it is necessary to map these heterogeneous structures into a unified merged structure in the preprocessing step, which facilitates high scalability and enables more efficient downstream procedures.

Challenge II: *How can the merged index achieve good search performance?* The merging process must preserve the structural properties that are crucial for efficient ANNS. The state-of-the-art (SOTA) graph-based methods (e.g., HNSW [37] and Vamana [22]) achieve high performance through careful edge selection and pruning strategies, which optimize the search path length and thus reduce distance computations. During index merging, we need to ensure the combined graph-based index maintains similar connectivity and navigability properties to those of the index directly constructed on the merged dataset to guarantee efficient search performance.

Solution and Contributions. We propose a *Fast Graph-based Indexes Merging (FGIM)* framework to effectively merge graph-based indexes for ANNS. FGIM reformulates the index merging problem as a graph merging problem by mapping disconnected graph-based indexes onto a unified graph structure and optimizing it for ANNS tasks. Specifically, our framework consists of three core steps: *PGs to k-NNG transformation*, *k-NNG refinement*, and *k-NNG to PG transformation*. Firstly, the *PGs to k-NNG transformation* extracts enough potential candidate neighbors from the original graph-based indexes. It retains the existing neighborhood relationships and elaborately selects pivotal cross-graph candidate neighbors through *cross-querying*, then maps multiple graphs into a unified *k-NNG* structure. In the next step, the obtained *k-NNG* is iteratively optimized through a streamlined *k-NNG refinement* process to establish more accurate direct connections between cross-graph vertices, incorporating *indegree-aware* mechanism to enhance graph connectivity.

Table 1. Symbols and Descriptions

Symbol	Description
$\mathcal{X}$	the base dataset
h	the number of indexes to be merged
$G(V, E)$	the graph index with vertex set V and edge set E
$\delta(\cdot, \cdot)$	the distance function
$\vec{x}, \vec{x}_b, \vec{x}_q$	a normal, base, and query vector
$ \cdot $	the cardinality of a set
L	the candidate pool size in search
k	the maximum outdegree of the graph
$\phi(\cdot)$	the id mapping function from old to new indexes
$C_i(u)$	the candidate neighbor set of vertex u in G_i .
$N_G(u)$	the neighbors of vertex u in graph G
$\mathcal{T}(u)$	the indegree of vertex u

Then, the neighbor selection and graph optimization processes are applied in the *k-NNG to PG Transformation* step, both designed to improve search performance and graph navigability. Besides, a merging strategy tailored for HNSW [37] is proposed to facilitate the integration of hierarchical structures. The FGIM framework produces a merged graph-based index optimized for ANNS tasks, ensuring both applicability and efficiency. To summarize, we make the following contributions:

- We formulate the graph-based index merging problem in §2, then propose a universal framework FGIM for merging graph-based indexes in §3, which is designed to be compatible with a wide range of mainstream graph-based ANNS methods.
- We present the implementation details of our proposed framework in §4, including the *PGs to k-NNG transformation*, *k-NNG refinement*, and *k-NNG to PG transformation*.
- Extensive experiments on real datasets show the efficiency and applicability of our proposed framework in §5.

2 Preliminaries

We show problem definitions in §2.1 and discuss the current studies in §2.2. Table 1 lists the frequently used notations in this paper.

2.1 Problem Definition

2.1.1 ANNS Definition. Let $d \in \mathbb{N}$ be the dimension of the vector, $n \in \mathbb{N}$ be the number of vectors in the dataset. The dataset is given by $\mathcal{X} = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_n\} \subset \mathbb{R}^d$. Let $\delta(a, b) : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ denote the distance function that computes the distance between any two vectors $\vec{x}_a$ and $\vec{x}_b$ in $\mathcal{X}$ in a specific metric space (e.g., ℓ_2 or cosine). The task of *k*-Nearest Neighbor Search (*k*-NNS) is defined as follows:

Definition 1 (*k*-NNS). Given a finite dataset $\mathcal{X}$ and a query vector $\vec{x}_q \in \mathbb{R}^d$, and a parameter $k \leq n$, *k*-NNS retrieve the set $\mathcal{R}$ consisting of the *k* vectors from $\mathcal{X}$ that have the minimum distance to $\vec{x}_q$ based on δ . For $\forall \vec{x}_r \in \mathcal{R}$ and $\forall \vec{x}_s \in \mathcal{X} \setminus \mathcal{R}$, we have $\delta(\vec{x}_q, \vec{x}_r) \leq \delta(\vec{x}_q, \vec{x}_s)$. $\mathcal{R}$ can be formally described as follows:

$$\mathcal{R} = \arg \min_{|\mathcal{R}|=k, \mathcal{R} \subset \mathcal{X}} \sum_{x \in \mathcal{R}} \delta(q, x) \quad (1)$$

In modern applications, the increasing size of datasets and the high dimensionality of vectors lead to significant amplification of the computational cost of exact *k*-NNS. To mitigate this, ANNS techniques construct optimized indexes that balance the search efficiency and result accuracy.

Algorithm 1: KNNSearch(q, G, L, k, ep)

Input: query vector $\vec{x}_q$, graph index $G = (V, E)$, search pool size L, k for top- k , optional enterpoint ep
Output: k nearest neighbors of $\vec{x}_q$

```

1 initialize  $C \leftarrow \{(ep, \delta(ep, q))\}$  and  $i \leftarrow 0$ 
2 while  $i < L$  do
3    $u \leftarrow C[i]$ 
4   mark  $u$  as visited
5   foreach  $v \in N(u)$  and  $v$  is not visited do
6      $\mid$  insert  $(v, \delta(v, q))$  into  $C$ 
7   sort  $C$  by  $\delta(q, x)$  and keep the top- $L$  results
8    $i \leftarrow$  index of the first unexpanded vertex in  $C$ 
9 return the first  $k$  results in  $C$ 

```

Definition 2 (k -ANNS). Given a finite dataset $\mathcal{X}$ and a query vector $\vec{x}_q \in \mathbb{R}^d$, ANNS constructs an Index I on $\mathcal{X}$. It then retrieves a subset C of $\mathcal{X}$ by I , and evaluates $\delta(\vec{x}_i, \vec{x}_q)$ to obtain the approximate k nearest neighbors $\tilde{\mathcal{R}}$ of q , where $\vec{x}_i \in C$.

The top- k recall rate ($Recall@k$) is commonly used to evaluate ANNS performance. Given a query $\vec{x}_q$ and a value $k \leq n$, $Recall@k$ is defined as:

$$Recall@k = \frac{|\mathcal{R} \cap \tilde{\mathcal{R}}|}{|k|} \quad (2)$$

Algorithm 1 shows a general search process of ANNS. Specifically, the algorithm maintains a candidate set C with search pool size $L (\geq k)$ (also referred to as *beam width*) to record the currently best L nearest neighbors of the query $\vec{x}_q$. The algorithm adds the enterpoint ep to the candidate set as the initialization of the candidate set (Line 1). The algorithm extracts the nearest neighbor from the candidate set at the beginning of each iteration (Lines 2-4), and then expands the selected vertex u by inserting all unvisited vertices $v \in N(u)$ into the candidate set (Lines 4-5). If the size of the candidate set exceeds L , the algorithm keeps the top- L nearest neighbors in the candidate set and removes the rest (Line 7). Afterward, the algorithm selects the next vertex to expand by discovering the first unvisited vertex in the candidate set at the end of each iteration (Line 8). The algorithm terminates when no unvisited vertex can be found in the C . Finally, the algorithm returns the top- k nearest neighbors in the candidate set as the search results (Line 9).

2.1.2 Graph-based Indexes. Graph-based methods are reported to achieve superior search performance in ANNS tasks [7, 33, 56]. They map base vectors into a graph space, and then construct a proximity graph (PG) to represent the similarity relationships between base vectors.

Definition 3 (Proximity Graph). The PG of $\mathcal{X}$ is a graph $G = (V, E)$ with the vertex set V and edge set E . Each vertex $v_i \in V$ corresponds to a vector $\vec{x}_i \in \mathcal{X}$. Each edge $e_{ij} \in E$ represents the proximity between vertices v_i and v_j , which is determined by a specified distance metric (e.g., ℓ_2). The neighbors of a vertex v in V are denoted as $N_G(v)$.

Definition 4. (k -Nearest Neighbor Graph). k -NNG $G = (V, E)$ connects each vertex $v_i \in V$ to its k nearest neighbors in dataset $\mathcal{X}$, where $|N_G(v_i)| = k$.

The graph-based methods employ various strategies for PG construction, including search-based approaches (e.g., HNSW [37] and Vamana [27]) and refinement-based techniques (e.g., NSG [22] and τ -MNG [45]). Specifically, the construction process typically involves obtaining a subset of vertices as candidate neighbor set C for each vertex v (referred to as *Candidate Neighbor Acquisition*

in the literature [56, 63]) to construct an approximate k -NNG, which can be achieved through search or other heuristics, and then applying a pruning strategy to select the final neighbors from C named *Neighbor Selection*. For example, NSG [22] first constructs an approximate k -NNG and then prunes it to produce the PG. Similarly, while Vamana [27] starts from a random graph, it obtains the k -NNG through search before performing two rounds of optimization. Previous studies [63] show that increasing parameters such as $ef_{construction}$ in HNSW or L in Vamana produces a more accurate k -NNG, leading to better PG quality and improved ANNS performance. Although higher-quality k -NNGs incur greater construction costs, it requires a balance between accuracy and efficiency.

A critical distinction among these methods lies in their pruning strategies, which play a crucial role in shaping the structural properties of the graph and directly influence search performance [56]. While some methods (e.g., HNSW) incorporate Relative Neighborhood Graph (RNG) [52], others (e.g., NSG and Vamana) apply Monotonic Relative Neighborhood Graph (MRNG) [22]. RNG preserves edges based on relative proximity, whereas MRNG adds a monotonicity constraint to ensure greedy-search navigability.

PGs have become a focal point in recent ANNS studies [33, 56, 63] due to their ability to effectively capture the local neighborhood structure of high-dimensional data, facilitating efficient and accurate search operations. By representing data points as vertices and establishing edges between nearby points based on a distance metric, PGs preserve essential proximity relationships. Their fundamental properties, such as sparsity, which reduces computational complexity, and connectivity, which ensures navigability, enable efficient graph traversal while minimizing distance computations. These characteristics make PGs effective for balancing speed and accuracy in large-scale retrieval tasks.

Then, we have the following problem definition:

Definition 5 (Merging Graph-based Indexes for ANNS). Given h datasets $\{\mathcal{X}_i \subset \mathbb{R}^d\}_{i=1}^h$, we have their h pre-built graph-based indexes $\{G_i = (V_i, E_i)\}_{i=1}^h$ using the same distance metric δ . Let $B(\mathcal{X}_i)$ denote the index construction cost of dataset $\mathcal{X}_i$, and $M(\cdot)$ the cost of merging multiple indexes. The merge process is formulated as a multi-objective optimization problem:

$$\max \left\{ B(\mathcal{X}) - M(G_1, \dots, G_h), Q(\hat{G}), R(\hat{G}) \right\},$$

where $\mathcal{X} = \bigcup_{i=1}^h \mathcal{X}_i$, the merged index is $\hat{G} = (V, E)$ with $V = \bigcup_{i=1}^h V_i$, $Q(\hat{G})$ is the query-per-second throughput of $\hat{G}$, and $R(\hat{G})$ is the recall rate of $\hat{G}$.

The goal of merging graph-based indexes for ANNS is to build an index $\hat{G}$ for the union set $\mathcal{X}$ of the h datasets with the maximum saving cost comparing with building $\hat{G}$ on $\mathcal{X}$ directly, while optimizing the query throughput $Q(\hat{G})$ and recall rate $R(\hat{G})$ of $\hat{G}$.

2.2 Current Studies

Apart from the brute-force reconstruction method, prior work in the literature can be summarized into the following three categories.

Incremental Indexing. Incremental insertion provides a practical approach to index merging, where vectors from smaller datasets are sequentially added to a base index built on a larger dataset. Some search-based structures, such as HNSW [37], naturally support this process through point-wise insertions. However, this approach has notable limitations. For instance, NSG [22] and Vamana [27] identify data centroids during preprocessing and use Algorithm 1 starting from these centroids to select candidate neighbors, ensuring a monotonic search path in the MRNG [22]. Incrementally inserting new points can shift centroids, potentially violating the MRNG property and requiring a full reconstruction of the index. Moreover, incremental insertion relies solely on the base index

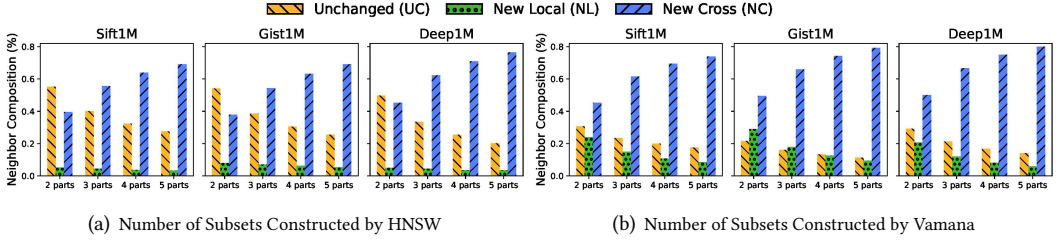


Fig. 3. The neighbor composition of HNSW and Vamana.

and ignores indexing information from other indexes, which limits its ability to fully leverage their potential contributions to the merged index.

***k*-NNG Merge.** Zhao et al. [67] propose a method for merging two *k*-NNGs, which first introduces random edges between the two disjoint *k*-NNGs, followed by applying NNDescend [19] as refinement. However, since this method is designed specifically for merging two *k*-NNGs, it is neither suitable for efficient ANNS tasks, according to recent studies [33, 56], nor compatible with mainstream graph-based indexing methods (e.g., HNSW [37], Vamana [27] and NSG [22]), which do not produce *k*-NNGs as their final index structures.

DiskANN. DiskANN [27] introduces a scalable index merging strategy to handle billion-scale datasets using multiple shards. The method partitions the base data into clusters via *k*-means and assigns each point to its ℓ nearest centers to form *k* partitions, then constructs a Vamana index for each partition. Finally, merging is performed by taking the union of the subgraphs. However, this approach entails repeated indexing on the same data point across partitions, inflating the index size by a factor of ℓ . Additionally, search performance becomes highly sensitive to the choice of clustering parameters. Since the method depends on spatial clustering of the original data and forms cross-partition connections through repeated indexing, it struggles to generalize to arbitrarily distributed data without dedicated preprocessing, resulting in substantial performance degradation.

Overall, the aforementioned methods exhibit limitations in terms of universality, efficiency, and effectiveness. These limitations highlight the potential to design a novel merging method that can effectively leverage the strengths of existing graph-based indexes while addressing the challenges of efficiency and search performance.

3 Fast Graph-based Indexes Merging Framework

We propose a general framework for merging graph-based ANNS indexes. We first introduce the motivation behind our solution in §3.1. Next, in §3.2, we explain why we convert the original indexes into a *k*-NNG. We then present the full pipeline in §3.3.

3.1 Motivation

The primary reason why existing methods are incompetent in merging graph-based indexes is that they do not fully utilize the neighborhood information from the existing indexes. Intuitively, connected vertices in the original indexes are likely to remain neighbors in the merged index. Even when certain neighbors cannot be retained as direct connections in the new graph due to outdegree constraints, where they are replaced by newly discovered closer vertices, they still provide valuable proximity information. This aligns with the fundamental principle in graph-based ANNS that a neighbor of a neighbor may also be a neighbor [19, 33, 56], which can be exploited to improve the merging process.

In Figure 3, we compare the neighbor composition of small indexes to be merged with that of the large index constructed on the full dataset. Three real-world datasets (i.e., Sift [28], Gist [28] and Deep [10]) are randomly and equally partitioned into h ($h = 2 \dots 5$) subsets, on each of which we

build a graph index $G_1, G_2, \dots, G_h$ using method M (i.e., HNSW [37] and Vamana [27]). Subsequently, we construct a graph index G' on the full dataset using the same indexing method M with identical parameters. For each vertex v in index G_i , we identify the corresponding vertex v' in G' such that $\vec{x}_v = \vec{x}_{v'}$. We then compare the differences in neighbor sets $N_{G_i}(v)$ and $N_{G'}(v')$, where the proportion of shared neighbors is denoted as π_{uc} , newly discovered intra-subgraph neighbors as π_{nl} , and newly discovered cross-subgraph neighbors as π_{nc} . We aggregate these values across all vertices to obtain the final results. Our observations reveal that a consistent portion of neighbors remains unchanged, with HNSW demonstrating higher π_{uc} (e.g., when using only two subgraphs, π_{uc} reaches 55.3%, 54.2%, and 49.6%, respectively). Moreover, among the newly added connections, cross-subgraph links consistently account for a larger share than intra-subgraph links, i.e., $\pi_{nc} > \pi_{nl}$. *These findings suggest that our method should aim to preserve original neighbor relationships while also promoting the establishment of effective cross-subgraph connections.*

3.2 Why Using k -NNG?

Another key challenge in merging multiple graph-based indexes arises from *structural heterogeneity*, which stems from variations in the number of neighbors each vertex maintains across different indexes. These discrepancies originate from the distinct *construction strategies* and *pruning techniques* employed by various indexing methods [33, 56]. For instance, HNSW [37] creates long edges to facilitate efficient routing, Vamana [27] preserves medium-to-long edges during pruning, and τ -MNG [45] retains certain short-edge connections. These strategies result in intrinsic differences in graph structure and connectivity, making merging challenging: directly combining graphs can lead to imbalanced connectivity, disconnected components, or suboptimal routing. As discussed in §2.1.2, mainstream graph construction methods typically follow a two-stage paradigm [56]: *Candidate Neighbor Acquisition*, which builds a k -NNG, followed by *Neighbor Selection*, which forms the final PG. This naturally raises the question: *why not adopt a similar pattern for merging existing PG indexes?* Specifically, we convert each input PG into a unified k -NNG, ensuring uniform vertex degrees and consistent connectivity across graphs. This transformation facilitates efficient and effective graph merging by (i) providing a standardized intermediate representation that preserves the core local neighborhood structures of the original PGs, (ii) recovering informative edges that may have been pruned during individual PG constructions (i.e., from k -NNG to PG) [63], as even edges omitted in all local indexes may still benefit the global index. By converting the PG back into a k -NNG, we no longer need to know the specific structural properties of the original graph-based index, thus mitigating heterogeneity across different indexes.

In summary, our approach preserves the existing neighborhood relationships from existing indexes and leverages this information to establish effective cross-graph connections, thereby pre-constructing a k -NNG that encodes the original graph structural information as the initial solution for merging.

3.3 Pipeline of the Framework

Figure 4 illustrates the FGIM pipeline, which consists of three steps: *PGs to k -NNG*, *k -NNG Refinement*, and *k -NNG to PG*. The *PGs to k -NNG* step plays a key role by generating *Candidate Neighbor Acquisition* results from existing graph-based indexes, which are then used as input to the *k -NNG Refinement* stage to obtain more accurate candidate neighbors. Finally, the *k -NNG to PG* step constructs the final merged graph-based index, improving both search performance and connectivity.

- *PGs to k -NNG transformation* (§4.1): The transformation from PGs to k -NNG consists of two techniques: *cross-querying* and *minimum querying strategy*. Given a set of PG $\mathcal{G} = \{G_1, G_2, \dots, G_h\}$,

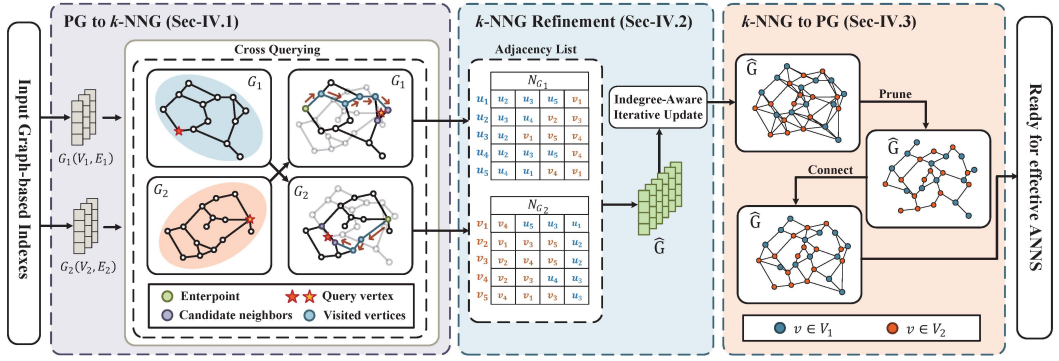


Fig. 4. The workflow of our FGIM framework with two graph-based indexes.

it first generates a candidate neighbor set $C_i(u)$ for each vertex u in G_i by efficiently extracting neighborhood relationships from the existing graph-based indexes. Then, the top- k nearest neighbors from $C_i(u)$ are selected to form the *Candidate Neighbor Acquisition* results, with each vertex obtaining k candidate neighbors. From a macroscopic perspective (i.e., the entire graph), this process can be viewed as a transformation from the existing PGs to an interconnected k -NNG, which serves as a bridge between the original graph-based indexes and the final merged index.

- **k -NNG Refinement (§4.2):** k -NNG refinement aims to refine the candidate neighbor set by identifying high-quality neighbors that may have been previously overlooked during the *PGs to k -NNG transformation* while eliminating redundant connections to ensure that k -NNG maintains high accuracy. Specifically, we apply an improved refinement method to discover closer neighbors, which iteratively updates the neighbors of each vertex u based on the current neighbor set $N_G(u)$. Moreover, to maintain graph connectivity, an indegree repair mechanism is integrated into the refinement process, enhancing the overall reachability.
- **k -NNG to PG transformation (§4.3):** Finally, we take above refined k -NNG as input to optimize the merged graph-based index with a *Neighbor Selection* process. Specifically, candidate neighbors are carefully selected to further improve the search performance while minimizing the negative impact on graph navigability, thus facilitating the graph-based ANNS. It is worth mentioning that different pruning strategies (e.g., RNG [52], MRNG [22], and τ -MNG [45]) can be applied to our method, depending on the specific requirements for the merged graph-based index.

4 Components of the FGIM Framework

4.1 PGs to k -NNG Transformation

In this subsection, we present the *PGs to k -NNG transformation* process, which constructs a preliminary merged graph structure in the form of a k -NNG from the existing graph-based indexes.

4.1.1 Candidate Neighbor Acquisition. We first identify the candidate neighbors from the existing graph-based indexes. To be formal, we denote the candidate neighbor set of vertex u in G_i as $C_i(u)$. Since we have a set of graphs $\{G_1, G_2, \dots, G_h\}$ with $n_1, n_2, \dots, n_h$ vertices, two types of candidate neighbors can be obtained: the *local candidate neighbors* $C_i^+(u)$ from the original graph G_i and the *cross candidate neighbors* $C_i^-(u)$ from the other graphs G_j ($j \neq i$).

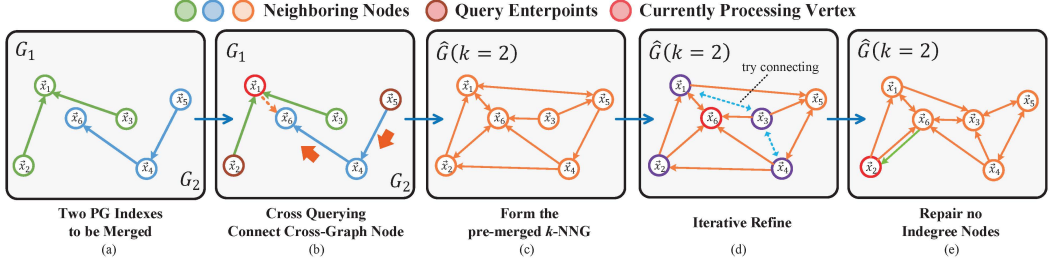


Fig. 5. An example of PGs to k -NNG transformation and k -NNG refinement.

Definition 6 (Local Candidate Neighbors). Given a vertex u in G_i in a set of graph-based indexes $\{G_1, G_2, \dots, G_h\}$, the local candidate neighbors can be obtained from the original graph G_i :

$$C_i^+(u) = \{v \in V_i \mid (u, v) \in E_i\} \quad (3)$$

where V_i and E_i are the vertex set and edge set of G_i , respectively.

To obtain local candidate neighbors, we retrieve the neighbors of vertex u directly from G_i . For cross candidate neighbors, we adopt a simple yet effective approach: treat each vertex u in G_i as a query point $\tilde{x}_u$ and perform Algorithm 1 in the other indexes $\{G_1, G_2, \dots, G_h\} \setminus G_i$. This *cross-querying* process yields candidate neighbors across different graph-based indexes.

Definition 7 (Cross Candidate Neighbors). Given a vertex u in G_i in a set of graph-based indexes $\{G_1, G_2, \dots, G_h\}$ with enterpoints $\{ep_1, ep_2, \dots, ep_h\}$, and a search pool size L (i.e., the *beam width* used in beam search), the cross candidate neighbors can be obtained by querying the other indexes $\{G_1, G_2, \dots, G_h\} \setminus G_i$:

$$C_i^-(u) = \bigcup_{j \neq i} \text{KNNSearch}(\tilde{x}_u, G_j, L, ep_j) \quad (4)$$

The *cross-querying* technique acquires cross candidate neighbors from other PGs. In Figure 5, starting from $\tilde{x}_1$ in G_1 , we select $\tilde{x}_5$ as the entry point in G_2 and follow the search path to obtain $\tilde{x}_6$ as a candidate neighbor, creating a cross-graph connection between $\tilde{x}_1$ and $\tilde{x}_6$. This process is repeated for all vertices in G_1 and G_2 , producing a set of cross-graph candidate neighbors for each vertex.

4.1.2 Minimum Querying Strategy. However, searching for cross-candidate neighbors in other indexes can be computationally expensive. For example, the query time complexity of SOTA methods such as HNSW and NSG is $O(dmn^{\frac{2}{d}} \log n)$ with a probability of at least $1 - (1/e)^{\frac{d}{4}(1-\frac{3}{e^2})}$ [22, 45, 56], where d is the cost of a single distance computation in a d -dimensional space, m is the number of neighbors evaluated at each vertex (i.e., the graph's outdegree constraint), and $n^{\frac{2}{d}} \log n$ represents the search path length (i.e., the number of hops). Given h indexes, since vertices do not need to query their own index, the total cost of the *cross-querying* technique is $O(dm \sum_{i=1}^h n_i \sum_{j \neq i} n_j^{\frac{2}{d}} \log n_j)$. Under typical conditions in the industry, where each index has roughly the same size ($n_i \approx n'$), this introduces a substantial h^2 term, as each index queries all others, yielding $h \times (h-1) \approx h^2$ pairwise interactions. To mitigate this, we propose a compensatory strategy: rather than retrieving precise neighbors at this stage, we quickly establish a set of inexpensive yet informative cross-graph connections. These serve as low-quality but *pivotal links* that provide essential cross-graph information, enabling efficient updates in subsequent iterative refinement.

The *Minimum Querying Strategy* ensures that each vertex u has at least k candidate neighbors in the initialized set. The minimum required search pool size L used in the *cross-querying* process

Algorithm 2: PGs to k -NNG Transformation

Input: a set of graph-based indexes $\mathcal{G} = \{G_1, G_2, \dots, G_n\}$, outdegree bound k , ID mapping function $\phi(\cdot)$

Output: k -NNG G

```

1 Initialize  $G \leftarrow \emptyset$ , candidate set  $C \leftarrow \emptyset$ 
2  $L \leftarrow \frac{k}{|\mathcal{G}|-1}$ 
3 foreach  $G_i \in \mathcal{G}$  do
4   foreach  $u \in V_i$  do
5      $C \leftarrow \{v \in V_i \mid (u, v) \in E_i\}$ 
6     foreach  $G_j \in \mathcal{G} \setminus \{G_i\}$  do
7        $C \leftarrow C \cup \text{KNNSearch}(\vec{x}_u, G_j, L, L, ep_j)$ 
8     sort  $C$  in ascending order of the distance to  $\vec{x}_u$ 
9     map each  $v \in C$  to global ID using  $\phi(\cdot)$ 
10     $N_G(\phi(u)) \leftarrow \text{resize}(C, k)$ 
11 return  $G$ 

```

must satisfy: $L \geq \left\lceil \frac{k - \min_i |C_i^+(u)|}{|\mathcal{G}|-1} \right\rceil$. In the worst case (i.e., vertex u has no outgoing neighbors in its own index), this simplifies to $L \geq \left\lceil \frac{k}{|\mathcal{G}|-1} \right\rceil$.

Previous studies [63] have shown that a larger search pool size L improves k -NNG accuracy, leading to better PG quality and ANNS performance. However, our approach uses minimal search parameters, making the routing process more susceptible to local optima [47]. This will reduce k -NNG accuracy and degrade the merged PG quality. To address this, we optimize the k -NNG accuracy via a refinement method to ensure a high-quality merged index (§4.2). Once the k -NNG is sufficiently accurate, the final PG is generated through *Neighbor Selection* (§4.3), completing the merging process.

4.1.3 Mapping onto k -NNG. We obtain the candidate neighbor set of vertex u in G_i by combining local and cross neighbors: $C_i(u) = C_i^+(u) \cup C_i^-(u)$. Given a parameter k (the maximum outdegree of the merged graph), candidates exceeding k are ranked by distance to u , and only the top- k are retained. Analogous to M in HNSW and R in Vamana, k controls the trade-off between accuracy and efficiency and can be set to match the outdegree of the original graphs.

Moreover, each sub-index maintains a local identifier space, typically assigning vertex IDs starting from zero. To avoid identifier conflicts during merging, these local spaces must be aligned. We define an *ID mapping function* $\phi(\cdot)$ that maps each valid local vertex ID in a sub-index to a unique global ID in the merged index. For each local vertex identifier u_{src} , the function is defined as:

$$\phi(u_{\text{src}}) \rightarrow \begin{cases} (\text{true}, u_{\text{dst}}), & \text{if } u_{\text{src}} \text{ is retained in the merged index,} \\ (\text{false}, -), & \text{otherwise,} \end{cases}$$

where the boolean flag indicates whether the source vertex is considered valid in the context of the merged index. A value of false denotes an invalid or obsolete vertex that should be excluded, while true associates u_{src} with a unique global identifier u_{dst} in the destination index. In practice, $\phi(\cdot)$ assigns disjoint global ID ranges to different sub-indexes by applying appropriate offsets to their local identifier spaces, while filtering out invalid vertices, thereby guaranteeing global uniqueness and conflict-free merging. Since $\phi(\cdot)$ is user-defined, it provides flexibility in ID management to accommodate system-specific requirements. For example, in systems [14, 42, 54] based on

Log-Structured Merge (LSM) trees [44], vertex validity can be determined by record existence or associated metadata such as timestamps.

Algorithm 2 outlines the *PG-to-kNNG transformation*. The algorithm first initializes and determines the search candidate size L using the *minimum querying strategy* (Lines 1-2). For each vertex u in each PG G_i , it retrieves *local* candidates from G_i and *cross* candidates from the other PGs (Lines 3-7). Finally, the top- k neighbors are selected and assigned to $N_G(\phi(u))$ in the k -NNG (Lines 8-10).

4.2 k -NNG Refinement

After *PGs to k -NNG transformation*, the candidate neighbors are retrieved from the existing graph-based indexes. However, the proposed *minimum querying strategy* presents a critical challenge: the strategy may cause the *cross-querying* process to become trapped in a local optimum [47], resulting in low-quality cross candidate neighbors in an inaccurate k -NNG. To address this issue, we propose an *Indegree-Aware k -NNG Refinement* to improve the quality of the merged graph by iteratively refining the obtained k -NNG with *streamlined refinement* process while improving the graph connectivity via an *indegree repair mechanism*.

4.2.1 Streamlined Refinement. Since we have already established a sufficient number of cross-graph connections in §4.1, these pivotal vertices can be leveraged to introduce more precise cross-graph links in the k -NNG, which is achieved through the refinement method of the k -NNG [19]. This refinement operates by iteratively performing two batched operations, *Visit* and *Update*, over the entire graph to progressively improve neighbor quality.

Specifically, the *Visit* procedure organizes the neighbors and reverse neighbors of each vertex into candidate sets of newly discovered and previously processed vertices, while maintaining corresponding reverse graphs. A flag-based mechanism is employed to distinguish new and old neighbors, reducing redundant distance computations without affecting the final k -NNG structure. Based on this classification, vertices are inserted into the candidate sets C_n and C_o , and reverse links are recorded accordingly.

Given the candidate sets produced by *Visit*, the *Update* procedure examines all valid vertex pairs drawn from $C_n \times C_n$ and $C_n \times C_o$, where the two vertices originate from different graph indexes. For each such pair, the pairwise distance is computed and compared against the current farthest neighbors in their respective k -NN lists. If the new distance is smaller, the farthest neighbor is replaced, thereby establishing a direct and more accurate cross-graph connection. By iteratively applying this mutual update process, neighboring vertices gradually recognize each other, resulting in a k -NNG of higher quality.

Then, we first introduce a *cache-friendly merging strategy* based on a DFS-style traversal. In conventional approaches [19], vertices are accessed in ascending ID order, causing random memory access and poor cache efficiency. To address this, we adopt a DFS-style access pattern, where after processing a vertex, we immediately visit its neighbors to perform the *Update* operation. Following the ANNS principle that “a neighbor of a neighbor may also be a neighbor” [56], this leverages spatial locality, so data already in cache (for memory indexes) or in memory (for SSD-based indexes) can be reused, thereby reducing access overhead and speeding up merging.

Additionally, we propose a *streamlined refinement* approach that combines *Visit* and *Update* into a single pass per vertex. This eliminates the need for separate passes for visiting neighbors and updating, as required in conventional methods [19]. By integrating both operations within one traversal, we reduce overhead, access the latest graph snapshot, and continuously improve graph quality, ultimately enhancing the efficiency of the refinement process.

With the two optimizations, the refinement process converges quickly, as confirmed in our experiments (§5.5). Given a k -NNG G , we initialize two auxiliary reverse graphs, $\bar{G}_n$ and $\bar{G}_o$, to

Algorithm 3: k -NNG Refinement**Input:** k -NNG G , maximum iteration I_{max} **Output:** refined k -NNG G

```

1 Initialize the reverse graph  $\bar{G}_n, \bar{G}_o \leftarrow \emptyset$ 
2 repeat
3    $\mathcal{V}, \mathcal{S} \leftarrow \emptyset$ 
4   foreach  $u \in V, u \notin \mathcal{V}$  do
5     push  $u$  into  $\mathcal{S}$  and  $\mathcal{V}$ 
6     while  $\mathcal{S} \neq \emptyset$  do
7        $v \leftarrow \text{pop from } \mathcal{S}$ 
8        $C_n, C_o \leftarrow \text{perform Visit for } v$ 
9       push  $\forall w \in N(v)$  into  $\mathcal{S}$  and  $\mathcal{V}$  if  $w \notin \mathcal{V}$ 
10      move  $\forall v_1 \in \bar{G}_n[v]$  into  $C_n, \forall v_2 \in \bar{G}_o[v]$  into  $C_o$ 
11      perform Update on all  $(v_1, v_2) \in C_n^2 \cup (C_n \times C_o)$  s.t.  $v_1 \neq v_2, v_1, v_2$  from different  $G_i$ 
12    perform Indegree Repair for  $G$ 
13     $iter \leftarrow iter + 1$ 
14 until  $iter = I_{max}$ 
15 return  $G$ 

```

Algorithm 4: Indegree Repair Heuristic**Input:** graph G **Output:** repaired G , indegree recorder $\mathcal{T}$ of G

```

1  $\mathcal{T} \leftarrow \text{calculate indegrees for } \forall u \in V$ 
2 foreach  $\forall v^* \in V$  where  $\mathcal{T}(v^*) = 0$  do
3   repeat
4      $v \leftarrow \text{the closest unvisited neighbor in } N_G(v^*)$ 
5     foreach  $\forall v' \in N(v)$  in descending order of the distance to  $\vec{x}_v$  do
6       if  $v'$  is not replaced and  $\mathcal{T}(v') > 1$  then
7         Replace  $v'$  with  $v^*$  in  $N_G(v)$ 
8         Mark  $v^*$  in  $N_G(v)$  as replaced
9          $\mathcal{T}(v') \leftarrow \mathcal{T}(v') - 1$ 
10         $\mathcal{T}(v^*) \leftarrow \mathcal{T}(v^*) + 1$ 
11        break
12   until  $\mathcal{T}(v^*) > 0$  or all neighbors in  $N_G(v^*)$  are visited
13 return  $G, \mathcal{T}$ 

```

track neighbors labeled as *new* and *old*, respectively (Algorithm 3 Line 1). The refinement proceeds for up to I_{max} iterations. In each iteration, vertices are traversed in DFS order using a stack $\mathcal{S}$ and a visited set $\mathcal{V}$ (Lines 3-6). For each vertex v popped from the stack, the *Visit* operation accesses its neighbors $N_G(v)$ and classifies them into C_n (new) and C_o (old) (Lines 7-8). Reverse neighbors recorded in $\bar{G}_n$ and $\bar{G}_o$ are merged into these sets (Line 10). We then perform the *Update* operation on all valid *cross-graph* vertex pairs in $C_n^2 \cup (C_n \times C_o)$ (Line 11), ensuring pairs come from different subgraphs.

4.2.2 Indegree Repair Mechanism. k -NNG does not inherently guarantee full connectivity or reachability among all vertices [41]. For instance, a vertex with no indegree cannot be accessed during

the query process, rendering it unreachable. Moreover, such vertices are unlikely to participate in the iterative process, which can degrade the overall graph quality. To address this issue, we propose a *connectivity enhancement heuristic* that links such disconnected vertices, denoted as v^* , to the current graph G . Specifically, we try to connect each vertex v^* to its nearest neighbor $v \in N_G(v^*)$. To ensure that other vertices with low indegrees are not adversely affected, we traverse the neighbors of v to identify replaceable vertices that have not yet been replaced (i.e., not marked as *replaced*) and have an indegree greater than 1 (Line 6 in Algorithm 4). If v' in $N_G(v)$ meets these conditions, we replace v' with v^* in $N_G(v)$, and mark v^* as *replaced* in $N_G(v)$ (Lines 7-11). Otherwise, we continue to visit the next closest neighbor $v \in N_G(v^*)$, following the ascending order of distance to $\vec{x}_{v^*}$, until no neighbors are left. Importantly, these supplementary edges maintain the outdegree constraint while significantly reducing the number of connected components, thus improving reachability to disconnected vertices for query routing.

4.3 k -NNG to PG Transformation

Despite the refined k -NNG being already of high quality, it still falls short of performing ANNS tasks with both precision and efficiency because of weak navigability and longer search paths [33, 56], compared to the SOTA graph-based ANNS methods [22, 27, 37]. To bridge this gap, we propose a process to transform k -NNG back into PG, which consists of two main steps: 1) *neighbor selection*, 2) *connectivity enhancement*, and 3) *HNSW-adaptive merging*. Algorithm 5 shows the process of the k -NNG to PG transformation.

4.3.1 Neighbor Selection. Given a k -NNG G , we perform *neighbor selection* for each vertex $u \in V$. For each u , neighbors in $N_G(u)$ are sorted by distance to $\vec{x}_u$, and up to k neighbors are selected (Lines 2–8). A neighbor is kept if it is closer to u than to any previously selected neighbor (Lines 4–6), ensuring local diversity. This criterion is effective under common distance metrics (e.g., Euclidean, cosine) and can be extended to MIPS using standard ℓ_2 transformations [15]. To maintain reachability, we also ensure that pruning does not remove the only incoming edge of any vertex (Line 5).

This selection procedure removes distant neighbors that have closer alternatives in the graph space, establishing a sparser graph that enhances efficiency in ANNS tasks. Furthermore, the framework supports the integration of alternative pruning strategies (e.g., α -RNG [27] and τ -MNG [45]) by reimplementing the *neighbor selection* process, allowing for flexibility in the choice of methods.

4.3.2 Connectivity Enhancement. Although limiting vertex outdegree simplifies graph management, it can weaken global connectivity. To reduce strongly connected components, we perform *connectivity enhancement* on the pruned graph $\hat{G}$. Following the principle that *one tends to regard as important those who regard oneself as important* [43], we add reverse neighbors to each vertex's neighbor set $N_{\hat{G}}(u)$, then sort $N_{\hat{G}}(u)$ by distance to $\vec{x}_u$ (Lines 9–11). Finally, we retain only the top- k neighbors to avoid hubness (Line 12). These added edges improve long-range routing and graph navigability [47] while making better use of the degree budget.

We validate the effectiveness of the proposed k -NNG refinement (§4.2) and k -NNG to PG transformation (§4.3) on the Sift1M and Gist1M datasets. Figure 6 demonstrates that both components effectively improve ANNS performance by enhancing graph quality.

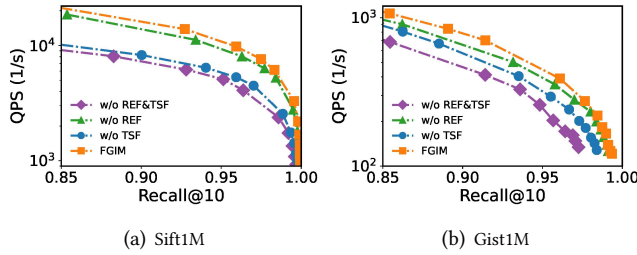
4.3.3 HNSW-Adaptive Merging. Many mainstream graph-based ANNS methods typically construct a single-layer index, where we can directly apply our FGIM framework to merge the indexes. However, the widely used HNSW [37] method builds a multi-layer graph structure, where each layer contains vertices of varying density, thereby enabling superior search performance. Notably,

Algorithm 5: k -NNG to PG Transformation**Input:** k -NNG G , indegree recorder $\mathcal{T}$, outdegree bound k **Output:** merged graph index $\hat{G}$

```

1 Initialize the graph  $\hat{G}(V', E') \leftarrow \emptyset$ 
2 foreach  $\forall u \in V$  do
3    $\mathcal{R} \leftarrow \emptyset$ 
4   foreach  $\forall v \in N_G(u)$  in ascending order of  $\delta(u, v)$  do
5     if  $\mathcal{T}(v) = 1$  or  $\forall w \in \mathcal{R}, \delta(u, v) < \delta(w, v)$  then
6        $\mathcal{R} \leftarrow \mathcal{R} \cup \{v\}$ 
7     break if  $|\mathcal{R}| = k$ 
8    $N_{\hat{G}}(u) \leftarrow \mathcal{R}$ 
9 foreach  $\forall u \in V'$  do
10    $N_{\hat{G}}(u) \leftarrow N_{\hat{G}}(u) \cup \{v \in V' \mid (v, u) \in E'\}$ 
11   sort  $N_{\hat{G}}(u)$  in ascending order of the distance to  $\vec{x}_u$ 
12   resize  $N_{\hat{G}}(u)$  to  $k$ 
13 return  $\hat{G}$ 

```

Fig. 6. Effectiveness of k -NNG refinement (REF, §4.2) and k -NNG to PG transformation (TSF, §4.3).

like skip lists [48], the number of vertices in each layer of HNSW decreases exponentially with increasing layer depth, i.e., $P(\text{level} \geq k) \propto \exp(-k)$. In real-world applications, this indicates that the number of non-base layer vertices is significantly smaller than that of the base layer. Therefore, we tried a simplistic yet effective strategy that first merges the base layer of HNSW using the proposed FGIM framework and subsequently reconstructs the higher layers accordingly.

Specifically, we reconstruct the upper layers following the original HNSW design principles. Each vertex in the merged base graph is first assigned a random level according to the HNSW level distribution, and the hierarchical graph-based index is initialized with a number of layers equal to the maximum assigned level. For a vertex with level $l > 1$, we perform greedy navigation to the closest vertex at level $l+1$, and then search for candidate neighbors from level l down to level 1 using Algorithm 1. At each level, candidate neighbors are pruned using the RNG pruning heuristic [52] introduced in HNSW [37], ensuring that the outdegree constraint is satisfied. Bidirectional edges are then added, and the same pruning strategy is applied to affected neighbors.

4.4 Complexity Analysis

In this part, we analyze the complexity of our FGIM framework.

PGs to k -NNG transformation. The complexity of this phase depends on the search complexity of the underlying graph-based index. For SOTA methods such as HNSW [37] and NSG [22], the expected search complexity is $O(dmn^{\frac{2}{d}} \log n)$ with probability at least $1 - (1/e)^{\frac{d}{4}(1-\frac{3}{e^2})}$ [45, 56],

where d is the cost of distance computation, m is the outdegree (i.e., the number of neighbors visited per vertex), and $n^{\frac{2}{d}} \log n$ corresponds to the search path length. For the merged graph, we set the outdegree constraint k equal to that of the original graphs (i.e., $k = m$). Under our *Minimum Querying Strategy*, only $m' = \lceil k/(h-1) \rceil$ neighbors are visited at each hop, where $m' < m$. Given h PGs $\mathcal{G} = \{G_1, G_2, \dots, G_h\}$ with $\{n_1, n_2, \dots, n_h\}$ vertices, the complexity of this phase is $O(\frac{kd}{h-1} \sum_{i=1}^h n_i \sum_{j \neq i} n_j^{\frac{2}{d}} \log n_j)$. When all $n_i = n'$, this simplifies to $O(kdhn'^{\frac{2+d}{d}} \log n')$, which provides a good approximation for the general case.

k-NNG refinement. This step involves at most $I_{\max}$ iterations, where each iteration refines the graph using the *streamlined refinement* technique with complexity $O(I_{\max} \cdot n \cdot k^2 d)$. Additionally, the *indegree repair mechanism* incurs a complexity of $O(I_{\max} \cdot n \cdot k)$, which is subsumed by the dominant term. Therefore, the total complexity of the *k*-NNG refinement phase is $O(I_{\max} \cdot n \cdot k^2 d)$.

k-NNG to PG transformation. In this phase, the complexity of *neighbor selection* is $O(n \cdot k^2 d)$, while for *connectivity enhancement* is $O(n \cdot k)$. Hence, the total complexity of this phase is $O(n \cdot k^2 d)$. Combining both the *k*-NNG refinement and the PG transformation phases, the overall complexity becomes $O(I_{\max} \cdot n \cdot k^2 d)$.

HNSW-adaptive merging. The reconstruction of the upper-layer graph follows the original HNSW design, where elements are inserted sequentially via greedy search with $O(\log n)$ complexity per insertion [37, 56]. The probability that a node is assigned to level l or higher is given by $\mathbb{P}(\text{level}_i \geq l) = \int_0^{e^{-l \cdot \log(k)}} dx = k^{-l}$, which implies that the expected number of nodes in non-zero layers is n/k . As a result, the time complexity of this procedure is $O(\frac{n}{k} \log \frac{n}{k})$. For space, the number of nodes at level l is $n_l = n \cdot k^{-l} (1 - 1/k)$, and each maintains $O(k \cdot (l+1))$ neighbors. Summing across levels, the total space requirement becomes $\sum_{l=0}^{\infty} n \cdot k^{-l} (1 - \frac{1}{k}) \cdot (l+1) \cdot k = O(n \cdot \frac{k^2}{k-1}) = O(nk)$.

Total complexity. Summarizing all components, the time complexity of FGIM framework is:

$$O(\frac{kd}{h-1} \sum_{i=1}^h n_i \sum_{j \neq i} n_j^{\frac{2}{d}} \log n_j + I_{\max} \cdot n \cdot k^2 d + \frac{n}{k} \log \frac{n}{k}).$$

The space complexity of the framework is $O(nk)$, accounting for the candidate neighbor set of each vertex in the merged graph, the two auxiliary reverse-neighbor graphs, the stack and visited set used for DFS traversal during refinement, and the upper-layer graphs. Since each vertex has at most k positive neighbors, the reverse graphs require $O(nk)$ space.

5 Experimental Study

In this section, we present experimental results of our FGIM framework on seven real-world datasets. Our evaluation seeks to answer the following research questions:

RQ1: How does FGIM compare to incremental methods in terms of merging efficiency and search performance? (§5.2)

RQ2: How does FGIM perform in merging multiple indexes? (§5.3)

RQ3: How does FGIM compare to other graph-based indexes, and is it applicable in scenarios involving deletion and overlap? (§5.4)

RQ4: How do different strategies contribute to FGIM? (§5.5)

RQ5: How does FGIM scale w.r.t. data size and thread count? (§5.6)

5.1 Experimental Settings

Datasets. The experiments are conducted on several popular benchmarking datasets. All of them are real-world datasets and have been widely used in the literature [7, 56]. The datasets cover

Table 2. The properties of the datasets.

Dataset	Dim	#Base	#Query	Metric
Sift1M [28]	128	1,000,000	10,000	Euclidean
Sift2M [28]	128	2,000,000	10,000	Euclidean
Sift5M [28]	128	5,000,000	10,000	Euclidean
Gist1M [28]	960	1,000,000	1,000	Euclidean
Deep1M [10]	96	1,000,000	10,000	cosine
GloVe [46]	100	1,183,513	10,000	cosine
MSong [12]	420	994,185	1,000	Euclidean
Crawl [3]	300	1,989,995	10,000	cosine
Internet Search [1]	768	9,991,307	172	Euclidean

Table 3. Parameter variations for different algorithms.

Algorithm	Parameter Settings
<i>FGIM</i>	<code>max_degree</code> $\in \{8, 12, 16, 20, 24, 28, 32, 36, 48, 64, 96, 128\}$
<i>NSW, HNSW</i>	<code>maximum_neighbors</code> $\in \{4, 8, 12, 16, 20, 24, 36, 48, 64, 96\}$, <code>ef_construction</code> = 200
<i>Vamana, FreshDiskANN</i>	<code>R</code> $\in \{8, 12, 16, 20, 24, 28, 32, 36, 40, 56, 64, 72, 80\}$, <code>L</code> = 200 <code>α</code> = 1.2
<i>τ-MNG</i>	<code>b</code> $\in \{50, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$, <code>h</code> = 200
<i>NNDescent</i>	<code>k</code> $\in \{8, 12, 16, 20, 24, 28, 32, 36, 48, 64, 96, 128\}$

various applications such as image (Sift [28], Gist [28], Deep [10]), text (Glove [46], Crawl [3], Internet Search [1]), and audio (MSong [12]). The datasets' properties are summarized in Table 2. **Compared algorithms.** We evaluate 9 representative graph-based ANNS methods: (1) HNSW [37], a widely used hierarchical graph index; (2) Vamana [27], which optimizes HNSW's neighbor selection for improved search performance; (3) τ -MNG [45], which constructs a monotonic neighborhood graph from an existing index; (4) NSW [36], a classical navigable small-world graph; (5) NNDescent [19], a representative k -NNG construction method; (6) NNMerge [67], a merging method for k -NNGs; (7) DiskANN [27], which introduces a merging strategy for large-scale datasets; (8) FreshDiskANN [50], a representative graph-based index designed for SSDs; and (9) Lucene [57], which implements a merging strategy for HNSW in its vector search system. We did not apply PQ [28] or SQ [2], as these are orthogonal techniques applicable to any index. All baseline methods use their original pruning strategies, while our FGIM employs the algorithm in Algorithm 5 for neighbor selection during merging.

Parameters. We follow the approach of benchmarking papers [7, 33, 56] by employing grid search to determine the optimal parameter values for each index, ensuring that the algorithm achieves its best search performance. For building efficiency evaluations with parameter variations, we report the specific configurations in Table 3. For both the subgraphs to be merged and the merged graph, we consistently use the same construction parameters.

Computing Environment. We implemented our method in C++11 and compiled the code using CMake 3.30.2 with GCC 11.4.0 as the compiler. Almost all experiments were conducted on a machine equipped with an Intel Core i7-12700H CPU and 32 GB of RAM, running WSL2 Ubuntu 22.04 as

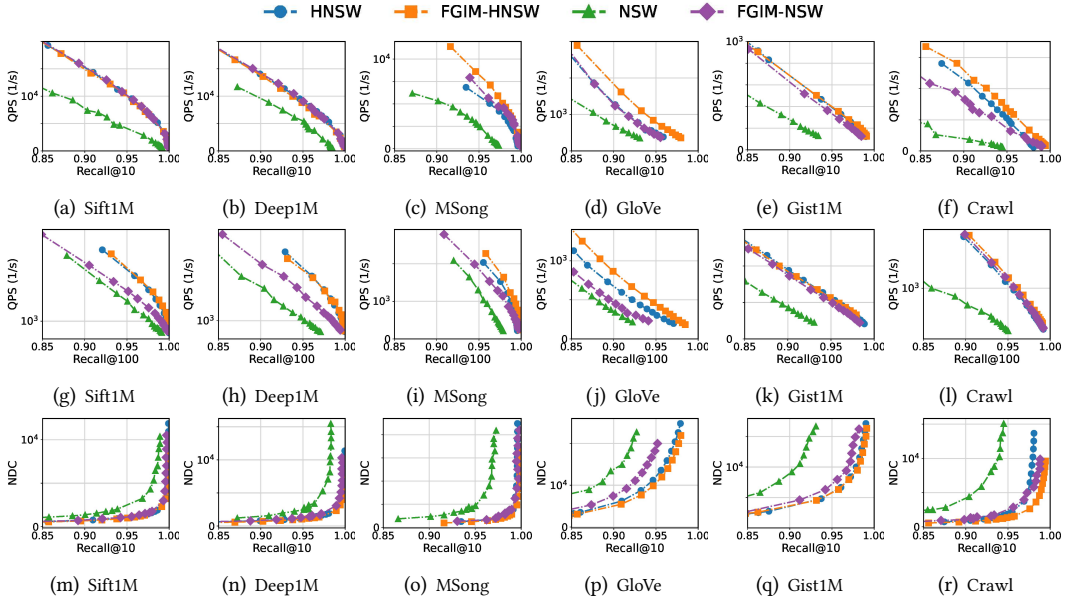


Fig. 7. Search performance of the merged index using FGM and incremental methods. (a–f: *Recall@10*-QPS, g–l: *Recall@100*-QPS, m–r: *Recall@10*-NDC; merging 2 indexes, Exp. 2)

the operating system. The only exception is that Exp.12 (§5.6) was conducted on a server equipped with an Intel Xeon Platinum 8163 CPU and 256 GB of RAM.

Measurements. To measure the accuracy of the search results, we use the *Recall* metric defined in §2. Additionally, Queries Per Second (QPS) and Number of Distance Computation (NDC) are used to measure the search efficiency. We measure the search performance of each method by plotting the QPS vs. *Recall@10* and *Recall@100* curves while varying the search parameters. Besides, the building efficiency is also evaluated by plotting the building time (BT) vs. *Recall@10* curves while varying the construction parameters.

5.2 Efficiency Evaluation

Exp.1 & 2: Comparison with the Incremental Method in Merging Efficiency and Search Performance. We evaluate the merge efficiency and search performance of our FGM framework against the incremental construction methods of HNSW, NSW, and FreshDiskANN. Each dataset is evenly split into two subsets, and separate indexes are built on them. For the baselines, the index on one subset is incrementally expanded by inserting all data from the other. HNSW and NSW are constructed single-threaded in memory, while FreshDiskANN operates in a disk-based setting using 20 threads. In contrast, FGM directly merges two pre-built indexes.

Figure 8 reports the construction cost of the merged index using the FGM framework and the incremental method of HNSW, NSW in the memory-based setting. Our methods achieve substantially higher efficiency, demonstrating speedups of 2.35× and 2.24× on Sift, 2.44× and 2.22× on Deep, 2.96× and 2.71× on MSong, 3.01× and 3.14× on GloVe, 3.51× and 2.79× on Gist, 2.16× and 1.66× on Crawl, respectively, compared to HNSW and NSW. We also report the build time of HNSW and NSW in Table 5 for reference. In the disk-based setting, Table 4 shows that FGM achieves speedups of 9.07×, 9.48×, 11.52×, 8.33×, 5.61×, and 5.87× over FreshDiskANN on the corresponding datasets. This improvement can be attributed to FGM’s ability to leverage both search mechanisms

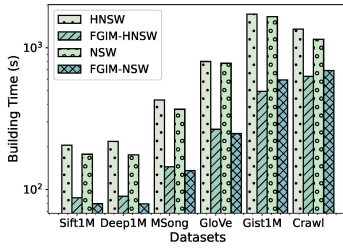


Fig. 8. Merging efficiency. (Exp. 1)

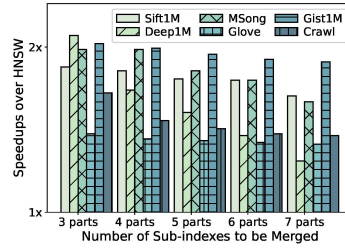


Fig. 9. Merging efficiency in multiple indexes (Exp. 4)

Table 4. Comparison of FreshDiskANN (FDA) and the proposed method in terms of merge time in the disk-based setting using 20 threads. (Exp. 1)

Methods	Sift1M	Deep1M	MSong	GloVe	Gist1M	Crawl
FDA	623.50	654.67	598.36	1588.05	965.59	1678.53
FGIM-FDA	68.78	69.08	51.92	190.67	172.18	286.11

Table 5. Comparison of build time (in seconds) for baseline methods in the memory-based setting. (Exp. 1)

Methods	Half Build Time			Total Build Time		
	Sift1M	Deep1M	MSong	Sift1M	Deep1M	MSong
HNSW	140.02	136.60	257.53	345.37	355.13	685.59
NSW	158.99	158.54	275.93	343.54	340.89	665.46

Table 6. Search performance (QPS) of FreshDiskANN (FDA) and our method in the disk-based setting. (Exp. 2)

Dataset	Recall@10 = 95%		Recall@10 = 97%		Recall@10 = 99%	
	FDA	FGIM-FDA	FDA	FGIM-FDA	FDA	FGIM-FDA
Sift1M	3841	3857	3054	2947	1571	1646
Deep1M	3840	3782	2894	2746	1584	1559
MSong	6709	6524	4999	4942	2146	2433

and iterative optimization strategies, thereby minimizing computational overhead. Overall, FGIM delivers higher efficiency than incremental methods in both in-memory and disk-based settings.

Figure 7 and Table 6 compare the search performance of incremental and our methods. Across all datasets, FGIM achieves comparable or better performance, showing that it can effectively merge indexes while preserving or improving search accuracy.

Exp.3: Comparison with other methods. As discussed in §2.2, many existing methods exhibit significant limitations. In this part, we evaluate these methods alongside our approach to demonstrate their shortcomings. Table 7 presents a comparison of different methods, focusing on merging time, *Recall@10* with fixed QPS (4000 for Sift, Deep, MSong, and 800 for GloVe, Gist, and Crawl), and index size. NNMerge achieves relatively fast merging but suffers from a lower *Recall@10*, consistent with findings in previous benchmark studies [33, 56], as *k*-NNG is not optimized for efficient ANNS. Moreover, the merging strategy employed by DiskANN leads to a noticeable drop in search accuracy, primarily due to the random spatial distribution of real-world datasets. Without a spatial clustering step during preprocessing, the merging process reduces to a naive structural union, which compromises global navigability and connectivity of the merged graph. Compared

Table 7. Comparison of existing strategies. (Exp. 3)

Method	Metric	Sift1M	Deep1M	MSong	GloVe	Gist1M	Crawl
HNSW	Indexing Time	3.42m	3.64m	7.13m	13.3m	28.8m	22.6m
	Recall@10	99.0%	98.7%	97.8%	93.9%	87.6%	96.9%
	Index Size	164.2M	173.2M	124.2M	292.2M	96.7M	348.0M
NNMerge	Indexing Time	3.24m	3.23m	6.31m	10.2m	28.3m	32.0m
	Recall@10	94.4%	92.6%	91.6%	89.9%	74.5%	90.6%
	Index Size	220.5M	220.6M	219.3M	522.5M	430.6M	925.6M
DiskANN	Indexing Time	–	–	–	–	–	–
	Recall@10	48.6%	49.7%	49.9%	48.7%	46.4%	49.7%
	Index Size	189.4M	210.0M	137.2M	430.0M	120.8M	474.4M
Lucene	Indexing Time	1.88m	2.13m	3.98m	5.37m	9.53m	10.1m
	Recall@10	98.8%	98.3%	97.6%	94.3%	86.7%	98.3%
	Index Size	119.3M	103.8M	78.8M	181.1M	66.2M	207.2M
FGIM (Ours)	Indexing Time	1.46m	1.49m	2.41m	4.43m	8.20m	10.5m
	Recall@10	99.3%	99.1%	98.8%	96.5%	91.3%	99.0%
	Index Size	123.3M	134.3M	92.3M	324.3M	100.7M	277.2M

to HNSW, Lucene’s merge strategy shows a modest decline in recall on several datasets, caused by reducing $ef_{construction}$ to improve merge efficiency at the cost of graph quality. In contrast, our FGIM framework outperforms other methods in terms of merging efficiency on most datasets, achieving consistently superior search performance. Overall, existing approaches remain ineffective for merging graph-based indexes.

5.3 Multiple Indexes Merging

As noted in §1, efficiently merging multiple graph-based indexes is crucial for real-time systems that generate frequent small indexes. We vary the number of indexes to merge from 3 to 7, each constructed by randomly selecting an equal number of vectors from the original dataset. We then compare our merging cost with HNSW’s incremental construction method.

Exp.4: Multi-Index Merging Efficiency. As shown in Figure 9, our merging method consistently outperforms the incremental approach as the number of indexes increases. Notably, although the acceleration gain of our method gradually diminishes with a growing number of indexes, it still achieves a noticeable speedup even when merging seven indexes. However, in real-world scenarios, the generated index fragments are merged offline in the background, preventing an excessive number of indexes. Therefore, our method remains effective in this context, ensuring its applicability and scalability to dynamic and evolving data environments.

Exp.5: Search Performance. Figure 10 reports the search performance of the merged index with the number of merged indexes ranging from 2 to 7. The results demonstrate that our method maintains consistent search performance across varying numbers of merged indexes. Across all datasets, the *Recall@10* drops by at most only ~1% under the same QPS as the number of merged indexes increases. This demonstrates that our method can effectively merge multiple indexes while preserving high search accuracy.

5.4 Framework Applicability

In this part, we apply FGIM to merge four representative graph-based indexes, i.e., Vamana, τ -MNG, NSW, and NNDescent, further exploring the generality and effectiveness of our framework.

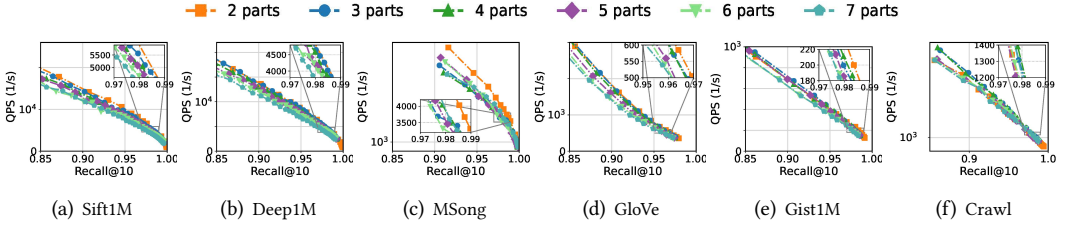


Fig. 10. Search performance of the merged index with the number of merged indexes ranging from 2 to 7. (a-f: *Recall@10*-QPS; Exp. 5)

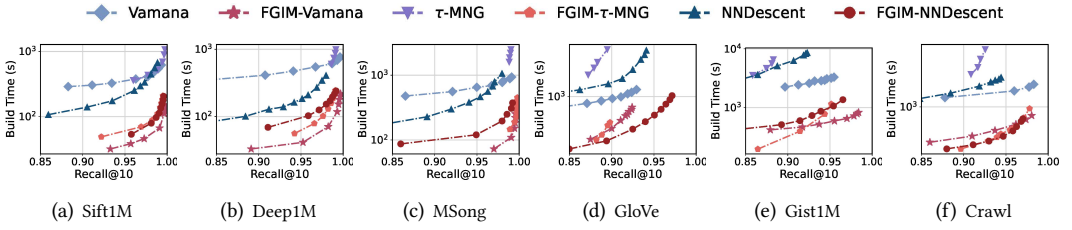


Fig. 11. Construction time for our merging methods and the original methods rebuilt from scratch. (a-f: *Recall@10*-BT; Exp. 6)

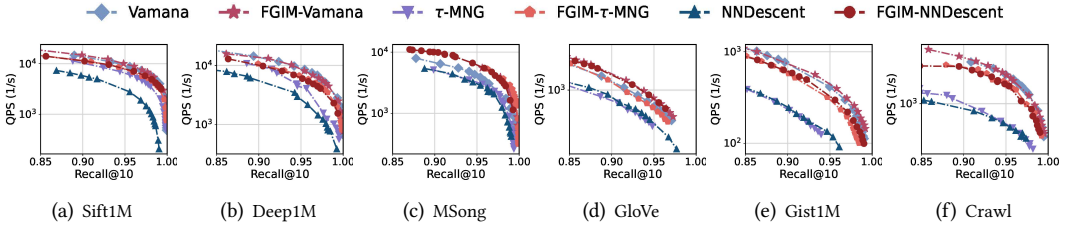


Fig. 12. Search performance for our merging methods and the original methods rebuilt from scratch. (a-f: *Recall@10*-QPS; Exp. 7)

Exp.6: Comparison with the Original Methods Rebuilt from Scratch. In this experiment, we employ building time and *Recall@10* curves that keep search parameters fixed (i.e., $L = 200$). For each method, we systematically vary its construction parameters to reconstruct indexes on the entire dataset from scratch, ensuring that each method has at least four different parameter configurations. Figure 11 presents the experimental results. We can find out that our methods fulfilled the merging task with high efficiency. Specifically, at the same *Recall@10*, (1) FGIM-Vamana achieved $3.4 \sim 6.9\times$ speedups over Vamana; (2) FGIM- τ -MNG achieved $4.3 \sim 23.2\times$ speedups over τ -MNG; (3) FGIM-NNDescent achieved $3.3 \sim 6.7\times$ speedups over NNDescent. On average, our method achieves a $7.4\times$ speedup, demonstrating its efficiency in merging graph-based indexes over reconstructing an entirely new index from scratch.

Exp.7: Search Performance. As depicted in Figure 12, FGIM-Vamana, FGIM- τ -MNG, FGIM-NSW, and FGIM-NNDescent consistently achieved comparable or superior performance compared to their original approaches. This improvement stems from the more accurate initialization of the k -NNG in our method. For instance, while Vamana starts from a randomly initialized graph, FGIM-Vamana begins with an approximate k -NNG obtained via the PG-to- k -NNG transformation. This better starting graph enables our method to identify more accurate k -nearest neighbors, improving graph quality and explaining FGIM-Vamana's superior performance on some datasets.

Table 8. Effectiveness of FGIM under the deletion and overlap settings on Sift1M. (Exp. 8)

Ratio	Deletion Setting		Overlap Setting	
	Recall@10	QPS	Recall@10	QPS
1/4	99.50%	3354	99.49%	3216
1/3	99.49%	3365	99.48%	3341
1/2	99.43%	3466	99.43%	3392

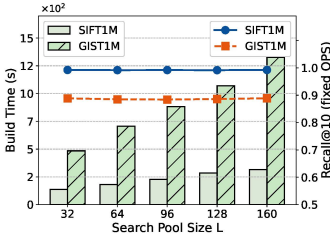


Fig. 13. Effects of search pool size. (Exp. 9)

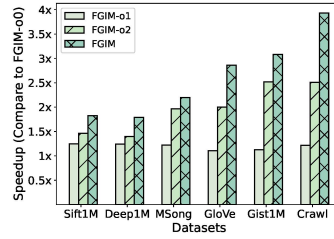


Fig. 14. Effects of acceleration techniques. (Exp. 10)

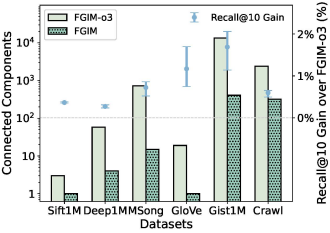


Fig. 15. Effects of indegree repair mechanism. (Exp. 11)

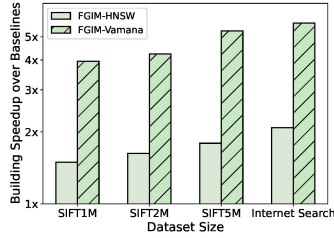


Fig. 16. Scalability study. (Exp. 12)

Exp.8: Deletion and Overlap Handling. In dynamic scenarios of modern vector database systems, it is essential to verify that FGIM can robustly handle deletions and overlapping data. To evaluate this, we conduct experiments on the Sift1M dataset. For each case, we build an index on the full dataset and another on a subset determined by a given ratio. In the deletion scenario, the subset represents deleted data, while in the overlap scenario, it overlaps with the existing data. As shown in Table 8, FGIM effectively manages both deletions and overlaps, maintaining high search accuracy and efficiency across different ratios. These results demonstrate the robustness of our framework in dynamic settings.

Overall, these findings demonstrate the high applicability of our methods, which are capable of efficiently and effectively merging graph-based indexes while preserving search accuracy.

5.5 Ablation Analysis

In this part, we assess the effects of our techniques used in the framework, i.e., the *Minimum Querying Strategy*, the *cross-querying*, the *streamlined refinement*, and the *indegree repair mechanism*.

Exp.9: Effects of Minimum Querying Strategy. According to our strategy, when selecting cross-graph candidate neighbors, we set the search pool size L to the minimum feasible value. A key question is whether this affects the quality of the graph. To investigate this, we increase L while keeping QPS fixed (4000 for Sift1M and 800 for Gist) and observe the changes in merging time and $Recall@10$, as shown in Figure 13. Our findings demonstrate that this strategy achieves the shortest index merging time without compromising graph quality, which can be explained that

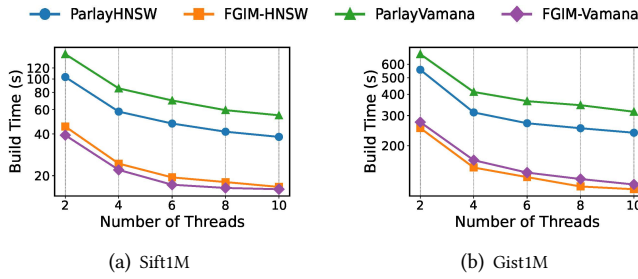


Fig. 17. Comparison of multi-threading efficiency. (Exp. 13)

the iterative refinement allows the graph structure to converge to a similar structure [19], thus mitigating the impact of initial candidate selection.

Exp.10: Effects of Optimization Technique. In this part, we evaluate the effects of the *cross-querying* and the *streamlined refinement*. We define four variants for comparison: FGIM-o0 without any technique; FGIM-o1, which incorporates only *cross-querying* and adopts the original refinement; FGIM-o2, which employs only *streamlined refinement* and initializes the merged index by randomly introducing vertices from the other indexes. We report the speedup achieved by each method over FGIM-o0 in Figure 14. Overall, each technique contributes to a significant acceleration of the merging process. Since these optimizations are mutually independent, their combined application FGIM yields the lowest construction cost.

Exp.11: Effects of Indegree Repair Mechanism. Figure 15 evaluates the effects of the *indegree repair mechanism*, reporting the Strongly Connected Components (SCC) in the merged index and the *Recall@10* improvement over the method without this mechanism (i.e., FGIM-o3). Overall, when this mechanism is applied, the number of strongly connected components decreases significantly, highlighting the effectiveness in improving graph connectivity within the degree budget. Besides, we observe that the *Recall@10* is improved across all datasets, demonstrating the effectiveness of this mechanism in enhancing search performance.

5.6 Scalability Study

Exp.12: Scalability Study. Figure 16 evaluates the scalability of our framework on the Sift and Internet Search dataset, with HNSW and Vamana as baselines. The results indicate that our method exhibits strong scalability as the number of vectors increases. Notably, the speedup achieved by our approach becomes amplified with larger dataset sizes, highlighting its superior efficiency in handling large-scale data. These findings demonstrate the robustness of our framework, making it well-suited for large-scale applications.

Exp.13: Multi-threaded Merging Efficiency. We use the parallel incremental insertion implementations of ParlayHNSW and ParlayVamana (based on ParlayANN) as baselines, and compare the merge efficiency of our framework under 2~10 threads, as shown in Figure 17. The results demonstrate that FGIM maintains high multi-thread efficiency across datasets and indexing methods, indicating that its efficiency can be further improved with increasing parallelism, while incurring only modest parallelization overhead.

6 Related Work

Recently, with the rise of *Large Language Models* (LLMs) and *retrieval-augmented generation* (RAG), ANNS [4, 61, 62] has attracted increasing attention, acting as a crucial component in these applications. Generally, ANNS methods are devoted to efficiently finding the most similar data points to a query in large-scale, high-dimensional datasets. Recent works [7, 40] have shown that graph-based

methods [22, 27, 37, 69] outperform traditional methods such as hashing-based methods [25, 26, 51], tree-based methods [11, 39, 49], inverted index-based methods [9, 10, 32, 66], and quantization-based methods [23, 28] in terms of search quality and efficiency.

The mainstream graph-based methods [13, 21, 22, 27, 36, 37, 45] statically build a proximity graph where each node is a base vector and edges connect several nearby vectors. Among them, HNSW [37] is a widely used graph-based method that is derived from small-world networks and is constructed incrementally in an online fashion. Other SOTA methods, such as NSG [22] and τ -MNG [45], construct indexes by performing refinement operations on the existing graph (e.g., refining a pre-constructed k -NNG). Vamana[27] achieves improved search performance by relaxing the neighbor selection policy to retain long-range edges. Due to their superior performance, these methods have been widely adopted in production [54, 60]. Our proposed framework is compatible with the abovementioned methods and can be adapted to merge their indexes.

To meet the demands of real-time systems in practical applications, several vector search systems [50, 58, 59, 65, 68] support dynamic updates. SPFresh [59] utilizes a clustering-based index but demonstrates diminished performance in high-dimensional vector spaces. FreshDiskANN[50], IP-DiskANN[58], and Greator [65] enable disk-based index updates. However, incremental methods fail to fully leverage the information from the existing indexes in index merging contexts, resulting in significant overhead.

7 Conclusion

In this paper, we study the problem of merging existing graph-based indexes into a single one for effective ANNS. We propose a general FGIM framework with three core techniques: 1) a *PGs to k -NNG transformation* that consists of *cross-querying* and *minimum querying Strategy* to extract initial candidate neighbors from the existing graph-based indexes; 2) a streamlined and indegree-aware *k -NNG refinement* method to improve candidate neighbors' quality; 3) a *k -NNG to PG transformation* and HNSW-adaptive process are presented for better ANNS performance. Experimental results demonstrate the generality and effectiveness of our FGIM framework, yielding noticeable speedups over SOTA methods without compromising the search performance.

8 Acknowledgment

This work was supported by Ant Group. Peng Cheng, Jingkuan Song and Heng Tao Shen were supported by Fundamental Research Funds for the Central Universities, Shanghai Central and Local Science and Technology Development Fund Project (Grant No. YDZX20253100002004). Yongxin Tong was partially supported by National Science Foundation of China (NSFC) (Grant Nos. 62425202, U21A20516, 62336003), the Beijing Natural Science Foundation (Z230001). Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China (NSFC) under Grant No. U22B2060, Guangdong-Hong Kong Technology Innovation Joint Funding Scheme Project No. 2024A0505040012, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Key Areas Special Project of Guangdong Provincial Universities 2024ZDZX1006, Guangdong Province Science and Technology Plan Project 2023A0505030011, Guangzhou municipality big data intelligence key lab, 2023A03J0012, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Zhujiang scholar program 2021JC02X170, Microsoft Research Asia Collaborative Research Grant, HKUST-Webank joint research lab and 2023 HKUST Shenzhen-Hong Kong Collaborative Innovation Institute Green Sustainability Special Fund from Shui On Xintiandi and the InnoSpace GBA. Xuemin Lin was supported by NSFC U2241211.

References

- [1] Ant Group 2025. *Alipay*. Ant Group. <https://www.antgroup.com>
- [2] Cecilia Aguerrebere, Ishwar Bhati, Mark Hildebrand, Mariano Tepper, and Ted Willke. 2023. Similarity search in the blink of an eye with compressed indices. *arXiv preprint arXiv:2304.04759* (2023).
- [3] Anon. Retrieved April 15, 2020. Common Crawl. <http://commoncrawl.org/>.
- [4] Sunil Arya and David M Mount. 1993. Approximate nearest neighbor queries in fixed dimensions.. In *SODA*, Vol. 93. Citeseer, 271–280.
- [5] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)* 45, 6 (1998), 891–923.
- [6] Akari Asai, Sewon Min, Zexuan Zhong, and Danqi Chen. 2023. Retrieval-based language models and applications. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 6: Tutorial Abstracts)*. 41–46.
- [7] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.
- [8] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-based vector search: An experimental evaluation of the state-of-the-art. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–31.
- [9] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.
- [10] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2055–2063.
- [11] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.
- [12] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.
- [13] Yuzheng Cai, Jiayang Shi, Yizhuo Chen, and Weiguo Zheng. 2024. Navigating labels and vectors: A unified approach to filtered approximate nearest neighbor search. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–27.
- [14] Cheng Chen, Chenzhe Jin, Yunan Zhang, Sasha Podolsky, Chun Wu, Szu-Po Wang, Eric Hanson, Zhou Sun, Robert Walzer, and Jianguo Wang. 2024. Singlestore-v: An integrated vector database system in singlestore. *Proceedings of the VLDB Endowment* 17, 12 (2024), 3772–3785.
- [15] Tingyang Chen, Cong Fu, Kun Wang, Xiangyu Ke, Yunjun Gao, Wenchao Zhou, Yabo Ni, and Anxiang Zeng. 2025. Maximum Inner Product is Query-Scaled Nearest Neighbor. *arXiv preprint arXiv:2503.06882* (2025).
- [16] Scott Cost and Steven Salzberg. 1993. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine learning* 10 (1993), 57–78.
- [17] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*. 271–280.
- [18] Magdalen Dobson, Zheqi Shen, Guy E Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. 2023. Scaling Graph-Based ANNS Algorithms to Billion-Size Datasets: A Comparative Analysis. *arXiv preprint arXiv:2305.04359* (2023).
- [19] Wei Dong, Charikar Moses, 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*. 577–586.
- [20] Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228* (2016).
- [21] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2021), 4139–4150.
- [22] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proceedings of the VLDB Endowment* 12, 5 (2017).
- [23] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.
- [24] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. 2009. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. 41–50.
- [25] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.
- [26] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

- [27] 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. *Advances in Neural Information Processing Systems* 32 (2019).
- [28] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.
- [29] Byungjun Lee, Kyung Hwan Oh, Hee Jung Park, Ung Mo Kim, and Hee Yong Youn. 2014. Resource Reallocation of Virtual Machine in Cloud Computing with MCDM Algorithm. In *2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. 470–477. <https://doi.org/10.1109/CyberC.2014.87>
- [30] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.
- [31] Qiang Li, Qiao Xiang, Yuxin Wang, Haohao Song, Ridi Wen, Wenhui Yao, Yuanyuan Dong, Shuqi Zhao, Shuo Huang, Zhaosheng Zhu, et al. 2023. More than capacity: Performance-oriented evolution of pangu in alibaba. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. 331–346.
- [32] Ruoxuan Li, Xiaoyao Zhong, Jiabao Jin, Peng Cheng, Wangze Ni, Lei Chen, Zhitao Shen, Wei Jia, Xiangyu Wang, Xuemin Lin, et al. 2025. SINDI: an Efficient Index for Approximate Maximum Inner Product Search on Sparse Vectors. *arXiv preprint arXiv:2509.08395* (2025).
- [33] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.
- [34] Jian Lin, Li Zha, and Zhiwei Xu. 2013. Consolidated cluster systems for data centers in the cloud age: a survey and analysis. *Frontiers of Computer Science* 7 (2013), 1–19.
- [35] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. 2024. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516* (2024).
- [36] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.
- [37] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.
- [38] Yitong Meng, Xinyan Dai, Xiao Yan, James Cheng, Weiwen Liu, Jun Guo, Benben Liao, and Guangyong Chen. 2020. Pmd: An optimal transportation-based user distance for recommender systems. In *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part II 42*. Springer, 272–280.
- [39] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence* 36, 11 (2014), 2227–2240.
- [40] Bilegsaikhan Naidan, Leonid Boytsov, and Eric Nyberg. 2015. Permutation Search Methods are Efficient, Yet Faster Search is Possible. *Proceedings of the VLDB Endowment* 8, 12 (2015).
- [41] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11, 1 (2002), 28–46.
- [42] Zhaojie Niu, Xinhui Tian, Xindong Peng, and Xing Chen. 2025. BlendHouse: A Cloud-Native Vector Database System in ByteHouse. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 4332–4345.
- [43] Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. 2024. Cagra: Highly parallel graph construction and approximate nearest neighbor search for gpus. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4236–4247.
- [44] Patrick O’Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O’Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33 (1996), 351–385.
- [45] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient approximate nearest neighbor search in multi-dimensional databases. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.
- [46] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.
- [47] Liudmila Prokhorenkova and Aleksandr Shekhovtsov. 2020. Graph-based nearest neighbor search: From practice to theory. In *International Conference on Machine Learning*. PMLR, 7803–7813.
- [48] William Pugh. 1990. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM* 33, 6 (1990), 668–676.
- [49] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.
- [50] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search. *arXiv preprint arXiv:2105.09613* (2021).

- [51] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proceedings of the VLDB Endowment* (2014).
- [52] Godfried T Toussaint. 1980. The relative neighbourhood graph of a finite planar set. *Pattern recognition* 12, 4 (1980), 261–268.
- [53] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-nn graph construction for visual descriptors. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1106–1113.
- [54] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.
- [55] Meng Wang, Weijie Fu, Xiangnan He, Shijie Hao, and Xindong Wu. 2020. A survey on large-scale machine learning. *IEEE Transactions on Knowledge and Data Engineering* 34, 6 (2020), 2574–2594.
- [56] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978.
- [57] Jasper Xian, Tommaso Teofili, Ronak Pradeep, and Jimmy Lin. 2024. Vector search with OpenAI embeddings: Lucene is all you need. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 1090–1093.
- [58] Haikexu, Magdalen Dobson Manohar, Philip A Bernstein, Badrish Chandramouli, Richard Wen, and Harsha Vardhan Simhadri. 2025. In-Place Updates of a Graph Index for Streaming Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2502.13826* (2025).
- [59] Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, et al. 2023. Spfresh: Incremental in-place update for billion-scale vector search. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 545–561.
- [60] Mingyu Yang, Wentao Li, Jiabao Jin, Xiaoyao Zhong, Xiangyu Wang, Zhitao Shen, Wei Jia, and Wei Wang. 2024. Effective and General Distance Computation for Approximate Nearest Neighbor Search. (2024). <https://arxiv.org/abs/2404.16322>
- [61] Mingyu Yang, Wentao Li, Jiabao Jin, Xiaoyao Zhong, Xiangyu Wang, Zhitao Shen, Wei Jia, and Wei Wang. 2025. Effective and general distance computation for approximate nearest neighbor search. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 1098–1110.
- [62] Mingyu Yang, Wentao Li, and Wei Wang. 2024. Fast High-dimensional Approximate Nearest Neighbor Search with Efficient Index Time and Space. *arXiv preprint arXiv:2411.06158* (2024).
- [63] 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. *Proceedings of the VLDB Endowment* 18 (2025), 1825–1838.
- [64] Kun Yu, Jiabao Jin, Xiaoyao Zhong, Peng Cheng, Lei Chen, Zhitao Shen, Jingkuan Song, Hengtao Shen, and Xuemin Lin. 2025. Approximate Nearest Neighbor Search of Large Scale Vectors on Distributed Storage. *arXiv preprint arXiv:2510.17326* (2025).
- [65] Song Yu, Shengyuan Lin, Shufeng Gong, Yongqing Xie, Ruicheng Liu, Yijie Zhou, Ji Sun, Yanfeng Zhang, Guoliang Li, and Ge Yu. 2025. A Topology-Aware Localized Update Strategy for Graph-Based ANN Index. *arXiv preprint arXiv:2503.00402* (2025).
- [66] Peitian Zhang and Zheng Liu. 2022. Bi-Phase Enhanced IVFPQ for Time-Efficient Ad-hoc Retrieval. *arXiv preprint arXiv:2210.05521* (2022).
- [67] Wan-Lei Zhao, Hui Wang, Peng-Cheng Lin, and Chong-Wah Ngo. 2021. On the Merge of k-NN Graph. *IEEE Transactions on Big Data* 8, 6 (2021), 1496–1510.
- [68] Shurui Zhong, Dingheng Mo, and Siqiang Luo. 2025. LSM-VEC: A Large-Scale Disk-Based System for Dynamic Vector Search. *arXiv preprint arXiv:2505.17152* (2025).
- [69] Xiaoyao Zhong, Jiabao Jin, Peng Cheng, Mingyu Yang, Lei Chen, Haoyang Li, Zhitao Shen, Xuemin Lin, Heng Tao Shen, and Jingkuan Song. 2025. EnhanceGraph: A Continuously Enhanced Graph-based Index for High-dimensional Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2506.13144* (2025).
- [70] Xiaoyao Zhong, Haotian Li, Jiabao Jin, Mingyu Yang, Deming Chu, Xiangyu Wang, Zhitao Shen, Wei Jia, George Gu, Yi Xie, et al. 2025. VSAG: An Optimized Search Framework for Graph-based Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2503.17911* (2025).
- [71] Chun Jiang Zhu, Tan Zhu, Haining Li, Jinbo Bi, and Minghu Song. 2019. Accelerating large-scale molecular similarity search through exploiting high performance computing. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 330–333.

Received July 2025; revised October 2025; accepted November 2025