Fast Graph-based Indexes Merging for Approximate Nearest Neighbor Search

Zekai Wu[1], Jiabao Jin[2], Peng Cheng[3], Xiaoyao Zhong[2], Lei Chen[4,5], Zhitao Shen[2], Jingkuan Song[3], Xiaofeng Cao[3], Heng Tao Shen[3], Xuemin Lin[6]
[1]East China Normal University, Shanghai, China; [2]Ant Group, Shanghai, China; [3]Tongji University, Shanghai, China; [4]HKUST (GZ), Guangzhou, China; [5]HKUST, Hong Kong SAR, China; [6]Shanghai Jiaotong University, Shanghai, China
zekaiwu@stu.ecnu.edu.cn; jinjiabao.jjb@antgroup.com; cspcheng@tongji.edu.cn; zhongxiaoyao.zxy@antgroup.com; leichen@cse.ust.hk; zhitao.szt@antgroup.com; jingkuan.song@gmail.com; xiaofeng.cao.uts@gmail.com; shenhengtao@hotmail.com; xuemin.lin@gmail.com

ABSTRACT

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 2.95× 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.

PVLDB Reference Format:
Zekai Wu, Jiabao Jin, Peng Cheng, Xiaoyao Zhong, Lei Chen, Zhitao She, Jingkuan Song, Xiaofeng Cao, Heng Tao Shen, Xuemin Lin. Fast Graph-based Indexes Merging for Approximate Nearest Neighbor Search. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Mingle-2012/pg-fast-merging.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Approximate Nearest Neighbor Search (ANNS) [2, 3] is a fundamental problem across various fields such as machine learning [11, 41], information retrieval [39, 48], recommendation systems [12, 29], vector databases [40], and large language models (LLMs) [4, 13]. Recent deep-learning methods have revolutionized data representation through embedding 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, thus indispensable for modern AI applications [4, 23, 26]. 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 [5, 24, 31, 42], graph-based indexes have emerged as one of the most effective techniques for ANNS due to their superior search accuracy.

Despite the strong performance of graph-based approaches, their practical deployment in industrial settings remains challenging. Real-world systems often operate in dynamic environments, where both data and device conditions change frequently. For graph-based methods, merging multiple indexes into a single one is commonly required in various scenarios, which poses significant challenges.

We first consider a distributed scenario illustrated in Figure 1. To handle large-scale data efficiently, distributed storage and indexing systems (e.g., Log-Structured Merge (LSM) trees [32]) have become increasingly prevalent. In such systems, each node constructs its index based on locally available data. As the distributed system evolves, scaling requirements may arise due to *cluster consolidation* [18, 25] or *resource reallocation* [35]. Specifically, computing nodes within a cluster are often merged to reduce costs and simplify management, necessitating the merging of existing indexes into a single one. Since certain methods [14, 16, 33] rely on dataset-oriented merging operations (i.e., rebuilding the entire index from scratch) due to their unique construction strategies, *this process incurs substantial computational overhead and limited scalability.*

Imagine another cost-effective and high-performance scenario for real-time industrial applications (e.g., *Retrieval-Augmented Generation* (RAG) [4, 23]), where indexing must be online performed to support continuous and efficient data writes. While Hierarchical Navigable Small World (HNSW) [28] 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 [21]), which trade query

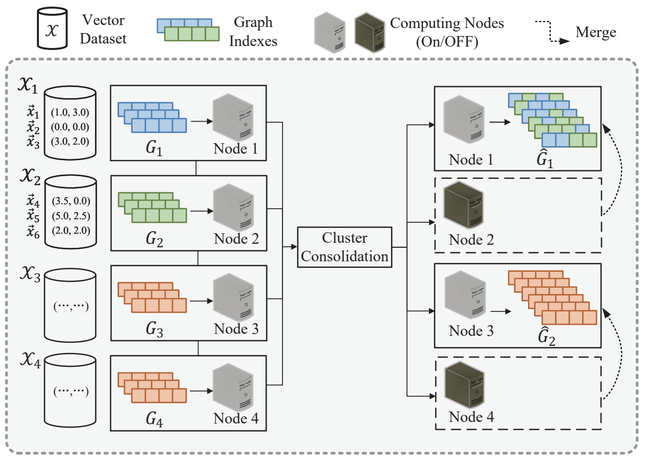


Figure 1: An illustration of cluster consolidation.

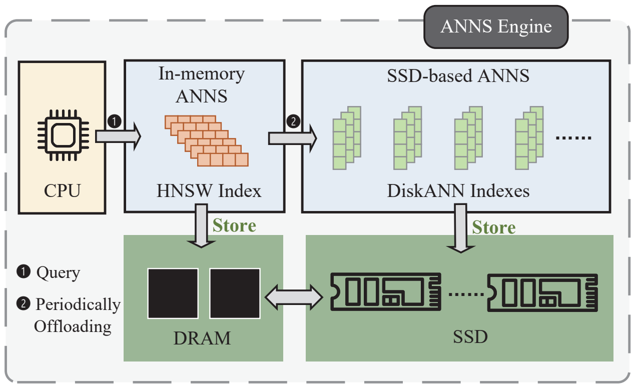


Figure 2: A hybrid engine for ANNS.

speed for cost efficiency. An industrial hybrid approach, shown in Figure 2, typically employs a size-constrained HNSW index (e.g., 1M~5M vectors) for real-time insertions, with periodic offloading to DiskANN. However, accumulating DiskANN indexes requires background reconstruction to preserve search quality, thereby imposing considerable computational costs. *This process is computationally expensive, making it difficult to be applied in large-scale online scenarios.*

Challenges. To the best of our knowledge, no existing research provides an efficient solution for merging graph indexes that meets the aforementioned requirements of industrial deployment. Therefore, we aim to develop a practical and effective approach to bridge this gap. Unlike prior studies [16, 21, 28, 33] on graph-based ANNS indexes, which primarily focus on the efficient and accurate construction of graph indexes from raw vector datasets, our work targets the efficient merging of multiple existing graph indexes into a single one. However, this problem is non-trivial, as two key challenges must be addressed.

Challenge I: *How can the original neighbor relationships be extracted from existing multiple indexes, facilitating the efficient construction of the merged graph-based index?* An effective merging approach should be able to extract valuable information (e.g., neighborhood relationship) from the original indexes, such as neighbor relationships between vertices. The challenge lies in aligning different ANNS indexing structures, as various methods employ different graph construction strategies. To ensure structural compatibility, it is essential to design a unified framework that can map these heterogeneous index structures onto a common merged index structure.

Challenge II: *How can the merged index maintain comparable search efficiency?* The merging process must preserve the structural properties crucial for efficient ANNS. State-of-the-art graph-based methods like HNSW [28] and NSG [16] achieve high performance through careful edge selection and pruning strategies that optimize the search path length and thus reduce distance computations. During index merging, we must ensure the combined graph maintains similar connectivity and navigability properties as would be obtained through direct reconstruction.

Solution and Contributions. We propose a Fast Graph-based Indexes Merging (FGIM) framework, an efficient framework for merging graph-based indexes to achieve effective ANNS. Our core idea is to reformulate 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, it consists of three core components: *PGs to k-NNG transformation*, *k-NNG refinement*, and *k-NNG to PG transformation*. First, the *PGs to k-NNG transformation* extracts enough potential candidate neighbors from the original graph-based indexes through *cross-querying* and *top-k selection*. Next, we propose a streamlined *k-NNG refinement* method that incorporates an *indegree-aware mechanism* to enhance graph connectivity, iteratively improving the quality of candidate neighbors while preserving graph connectivity. Third, in the *k-NNG to PG Transformation* phase, we apply the neighbor selection and graph optimization process, both designed to improve search performance and graph navigability. Finally, we propose a merging strategy tailored for merging the state-of-the-art HNSW [28] to facilitate the merging of hierarchical structures effectively. The FGIM framework produces a merged graph-based index that is well-optimized for ANNS tasks, ensuring both efficiency and effectiveness. To summarize, we make the following contributions:

- We formulate the graph-based index merging problem in §2.1, which aims to merge multiple graph-based indexes into a single one for effective ANNS.
- We propose a universal framework FGIM for merging graph-based indexes in §3, 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*.
- We integrate our FGIM framework with the state-of-the-art HNSW method. To achieve this, we propose a *HNSW-Adaptive Merging Strategy* in §5.
- Extensive experiments on real datasets show the efficiency and applicability of our proposed framework in §6.

Table 1: Symbols and Descriptions

Symbol	Description
$\mathcal{X}$	the base dataset
$G(V,E)$	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
$\lvert\cdot\rvert$	the cardinality of a set
L	the candidate pool size in search
m	the maximum degree of the graph
$C_i(u)$	the candidate neighbor set of vertex u in G_i.
$N_i(v),\overline{N}_i(v)$	the neighbors and reverse neighbors of vertex v in graph G_i

2 PRELIMINARIES

We show problem definitions in §2.1 and discuss the current studies in §??. 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 $\mathcal{X} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\} \subset \mathbb{R}^d$. Let $\delta(a,b) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ 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). The task of k-Nearest Neighbor Search (k-NNS) is defined as follows:

Definition 1. ***k-Nearest Neighbor Search (k-NNS).*** *Given a finite dataset $\mathcal{X}$ and a query vector $\vec{x}_q \in \mathbb{R}^d$, and a parameter $k \le n$, 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 $\forall \vec{x}_s \in \mathcal{X} \setminus R$, we have $\delta(\vec{x}_q, \vec{x}_r) \le \delta(\vec{x}_q, \vec{x}_s)$. $\mathcal{R}$ can be formally described as follows:*

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

In modern applications, the increasing size of datasets and the high dimensionality of vector representations significantly amplify the computational cost of exact k-NNS, making it impractical due to the curse of dimensionality [20]. To mitigate this, ANNS techniques construct optimized indexes that balance the trade-offs between search efficiency and result accuracy.

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 gets 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 $k \le n$, the recall rate is defined as:

$$Recall@k = \frac{|\mathcal{R} \cap \tilde{\mathcal{R}}|}{|k|} \tag{2}$$

Algorithm 1 details a general search process of ANNS. Specifically, the algorithm maintains a candidate set C with beam size $L(\ge k)$ 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

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

```
Input: query vector $\vec{x}_q$, graph index $G = (V, E)$, 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         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 $C$
```

keeps the top-L nearest neighbors in the candidate set and removes the rest (Lines 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 [42]. These methods map base vectors into a graph space, constructing a proximity graph (PG) to represent the similarity relationships between base vectors.

Definition 3. ***Proximity Graph (PG).*** *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 $N(v)$.*

State-of-the-art graph-based methods employ various strategies for PG construction, including search-based approaches (e.g., HNSW [28] and Vamana [21]) and refinement-based techniques (e.g., NSG [16] and τ-MNG [33]). Specifically, the construction process typically involves obtaining a subset of vertices as candidate neighbor set C for each vertex (referred to as *Candidate Neighbor Acquisition (CNA)* in the literature [42, 44]), which can be achieved through search or other heuristics, and then applying a pruning strategy to select the final neighbors from C (referred to as *Neighbor Selection (NS)*). 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 ANNS performance [42]. While some methods (e.g., HNSW) incorporate Relative Neighborhood Graph (RNG) pruning [38], others (e.g., NSG and Vamana) apply Monotonic Relative Neighborhood Graph (MRNG) pruning [16], leading to structural variations that affect efficiency and accuracy in ANNS tasks.

PGs have become a focal point in recent ANNS studies [24, 42, 44] 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.

Given all the preliminaries, we formally define the problem of merging graph-based indexes for effective ANNS.

Problem: *Let $\{G_i = (V_i, E_i)\}_{i=1}^{h}$ be h graph-based indexes, where each graph G_i is constructed from a dataset $\mathcal{X}_i$ using the same distance metric δ and the same indexing method M (e.g., HNSW, Vamana). Each dataset $\mathcal{X}_i$ consists of n_i vectors, all of the same dimensionality d. Our goal is to obtain a new merged graph-based index $\hat{G} = (V, E)$, satisfying the following properties:*

∘ *the vertex set is the union of all individual graph vertex sets $V = \bigcup_{i=1}^{h} V_i$, and the number of vertices in $\hat{G}$ is $n = \sum_{i=1}^{h} n_i$.*

∘ *the edge set E is constructed based on neighborhood relationships between the vertices in V.*

∘ *the merged graph $\hat{G}$ should be constructed time-efficiently, significantly faster than the time required to reconstruct the index from scratch.*

∘ *the merged graph $\hat{G}$ should be able to support efficient ANNS tasks, ensuring that the search performance is comparable to that of the index reconstructed from scratch.*

2.2 Current Studies

Apart from the brute-force reconstruction method, existing relevant studies can be summarized into the following three categories.

Incremental Indexing. Incremental insertion is a viable approach for merging indexes, where vectors from smaller datasets are sequentially inserted into the index of a larger dataset. Search-based construction methods like HNSW [28] inherently support this through point-by-point insertion. Nevertheless, this method presents noticeable limitations. First, it assumes that the underlying graph index supports incremental insertions, a property not inherently guaranteed in many graph-based methods (e.g., NSG [16] and Vamana [21]). Additionally, it relies exclusively on the base index during merging, neglecting the indexing information from other indexes, which limits its ability to fully leverage their potential contributions to the final merged index.

k-NNG Merge. Zhao et al. [47] proposes a method for merging two *k nearest neighbor graphs* (*k*-NNGs), which first introduces random edges between the two disjoint *k*-NNGs, followed by applying NNDescent [14] 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 [24, 42], nor compatible with mainstream graph-based indexing methods (e.g., HNSW [28], Vamana [21] and NSG [16]), which do not produce *k*-NNGs as their final index structures.

DiskANN. DiskANN [21] proposes a scalable index merging strategy to handle billion-scale datasets across different shards, which partitions data into overlapping clusters via *k*-means and constructs independent subgraphs for each cluster. However, the connections between subgraphs are not truly established, leading to a strong dependency on the choice of clustering parameters, which significantly impact search performance. Moreover, its reliance on clustered data preprocessing makes it inapplicable to arbitrary graph indexes in practice.

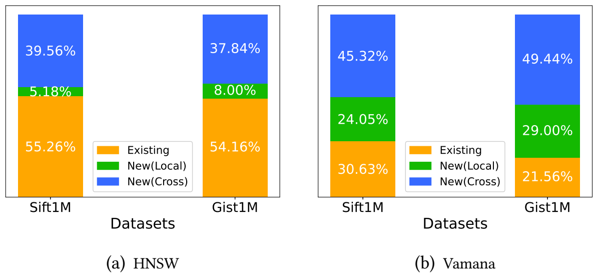


Figure 3: The neighbor composition of HNSW and Vamana.

Overall, the aforementioned methods exhibit limitations in terms of universality, efficiency, and effectiveness. These limitations highlight the substantial 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. Then, we describe the pipeline of our framework in §3.2.

3.1 Motivation

The primary reason why existing methods are incompetent in merging graph-based indexes is that they do not fully utilize the information (i.e., the neighborhood relationship) from the existing graph-based 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 out-degree constraints, where they are replaced by 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* [14, 24, 42], which can be exploited to enhance the effectiveness of the merging process.

To evaluate the significance of the information embedded in existing graph indexes, we partition the dataset into two subsets and construct separate indexes for each. We then apply the same indexing method to the entire dataset and examine whether a vertex's neighbors in the subgraphs are also present in its neighbor list in the full graph. As illustrated in Figure 3, a substantial proportion of neighbors in the full graph overlap with those in the subgraphs, with HNSW exhibiting a particularly high degree of overlap. Most new neighbors in the full graph originate from the other subgraph, while new connections within the same subgraph remain sparse. These results suggest that an effective merging strategy should preserve original neighborhood structures while establishing meaningful cross-graph connections.

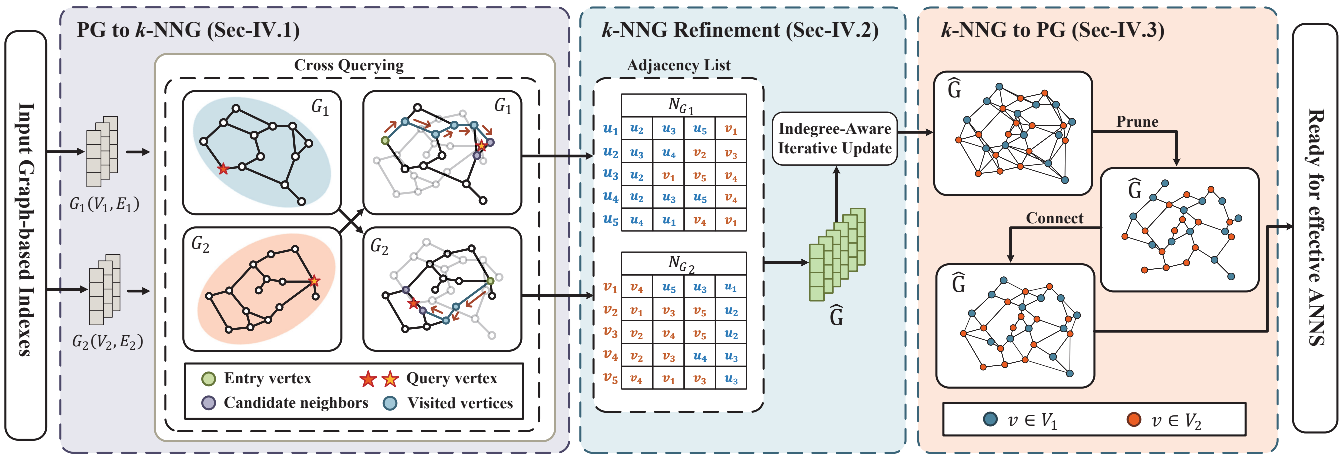


Figure 4: A pipeline of our FGIM framework with two graph-based indexes.

Another key challenge in index merging is the structural heterogeneity, which arises when directly merging PGs with differing degree distributions (e.g., HNSW [28] constructs a multi-layered graph with varying vertex degrees). To address this, we employ a k-NNG as a standardized intermediate representation, ensuring consistency in neighborhood structure across multiple PGs while preserving essential neighbor relationships from the original graph 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.

3.2 Pipeline of the Framework

Figure 4 illustrates the pipeline of the FGIM framework that contains three main steps: *PGs to k-NNG*, *k-NNG Refinement* and *k-NNG to PG*. The *k-NNG to PG transformation* plays a crucial role in the merging process, which forms the *CNA* results from the existing graph-based indexes. All *CNA* results are then regarded as input of the *k-NNG Refinement* step to obtain more accurate candidate neighbors. Finally, the *k-NNG to PG* transformation optimizes the merged graph-based index to improve search performance and graph connectivity.

PGs to k-NNG transformation (§4.1). The transformation from PGs to k-NNG consists of two steps: *cross-querying* and *top-k selection*. Given a set of PG $\mathcal{G} = \{G_1, G_2, \ldots, G_h\}$, it first generates candidate neighbor set $C_i(u)$ for each vertex v in G_i, by extracting neighborhood relationship from the existing graph-based indexes. Then, the top-k nearest neighbors from $C_i(u)$ are selected to form the *CNA* results. As a result, each vertex obtains 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. In this context, the k-NNG serves as a bridge between the original graph-based indexes and the final merged graph-based index, acting as an intermediate representation that facilitates the merging process.

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 each vertex retains exactly k neighbors. Specifically, we apply a fast refinement method to discover neighbors, which iteratively updates the neighbors of each vertex based on the current candidate neighbor set. Moreover, to maintain graph connectivity, a connectivity repair mechanism is integrated into the refinement process, ensuring the reachability of all vertices in the merged graph-based index.

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 *NS* 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 [38], MRNG [16] and τ-MNG [33]) 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

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 graph-based indexes $\{G_1, G_2, \ldots, G_h\}$ with $n_1, n_2, \ldots, n_h$ 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 $(j \neq i)$.

Definition 4. ***Local Candidate Neighbors.*** Given a vertex u in G_i in a set of graph-based indexes $\{G_1, G_2, \ldots, G_h\}$, the local candidate neighbors are defined as:

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

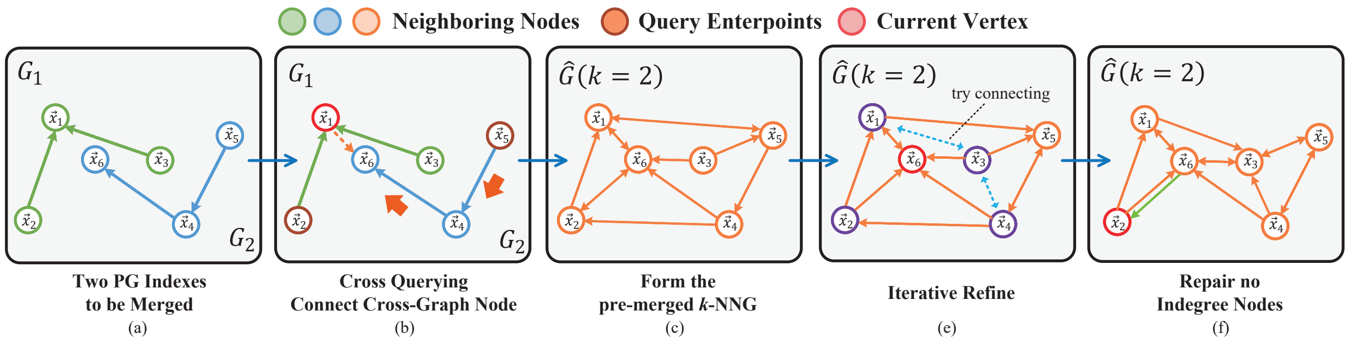


Figure 5: An example of *PGs to k-NNG transformation.*

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 the cross candidate neighbors, our idea is simplistic and straightforward: now that we have the indexes, we can directly query the other indexes to obtain the candidate neighbors. This process, referred to as *cross-querying*, involves treating each vertex u in G_i as a query point $\vec{x}_u$ and performing *KNNSearch* in the remaining indexes $\{G_1, G_2, \ldots, G_h\} \backslash G_i$ to acquire the cross candidate neighbors.

Definition 5. ***Cross Candidate Neighbors.*** Given a vertex u in G_i in a set of graph-based indexes $\{G_1, G_2, \ldots, G_h\}$ with enterpoints $\{ep_1, ep_2, \ldots, ep_h\}$, and a search pool size L, the cross candidate neighbors are obtained by querying the other indexes $\{G_1, G_2, \ldots, G_h\} \setminus G_i$:

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

We illustrate this process in Figure 5, where two PGs G_1 and G_2 are constructed from two datasets $\mathcal{X}_1$ and $\mathcal{X}_2$ shown in Figure 1, respectively. The *cross-querying* technique acquires candidate neighbors from the other PG. For example, starting from $\vec{x}_1$ in G_1, we choose $\vec{x}_5$ as the enterpoint in G_2. Following the search path, the candidate neighbor obtained from G_2 is $\vec{x}_6$, thereby establishing a cross-graph connection between $\vec{x}_1$ and $\vec{x}_6$. This process is repeated for all vertices in G_1 and G_2, resulting in a set of candidate neighbors for each vertex.

However, searching for the candidate neighbors in the other indexes is computationally expensive. For instance, the search cost of HNSW is $O(Ld \log n)$ [42], where d is the dimensionality of the dataset, n is the number of vertices in the index and L is the size of the search pool. Given h indexes, the total cost of the *cross-querying* technique is $O(Ld \sum_{i=1}^{h} n_i \sum_{j \neq i} \log n_j)$, since vertices do not need to query their own index, which is computationally expensive. To address this issue, we aim to minimize search time while ensuring feasibility, enabling the rapid acquisition of neighborhood relationship. Based on these premises, we provide Theorem 4.1 to show the minimum number of candidate neighbors required to ensure the feasibility of the merging process.

Theorem 4.1. *(Minimum Candidate Set Strategy for Merging) Let $\mathcal{G} = \{G_1, \ldots, G_n\}$ be a set of PGs, and let $C_i^+(u)$ and $C_i^-(u)$ denote the local and cross candidate neighbor sets for a vertex u in G_i. To guarantee that the merged candidate set $C_i^+(u) \cup C_i^-(u)$ contains at least k neighbors for all u, the minimum required cross candidate set size L must satisfy: $L \geq \left\lceil \frac{k - \min_i |C_i^+(u)|}{|\mathcal{G}|-1} \right\rceil$. In the worst case, this simplifies to $L \geq \lceil \frac{k}{|\mathcal{G}|-1} \rceil$.*

Proof Sketch. For each u, the cross candidate sets must compensate for any deficit in local candidates: $|C_i^-(u)| \geq k - |C_i^+(u)|$. Since cross candidates are aggregated from $|\mathcal{G}| - 1$ other graphs, the per-graph contribution must satisfy $L \geq \frac{k - |C_i^+(u)|}{|\mathcal{G}|-1}$. The worst-case bound follows when local candidates are empty. □

With the *Minimum Candidate Set* strategy, we can efficiently acquire candidate neighbors from the other indexes while ensuring feasibility. Although the quality of the obtained candidate neighbors may vary, the established cross-graph connections act as *pivotal links* for subsequent neighbor refinement. We will demonstrate how these initial candidates can be systematically optimized in §4.2. Finally, we obtain the candidate neighbor set of vertex u in G_i by combining the local and cross candidate neighbors $C_i(u) = C_i^+(u) \cup C_i^-(u)$.

4.1.2 Top-k Selection. *Top-k selection* is performed to obtain the final candidate neighbor set for each vertex. Given a predefined parameter k representing the maximum out-degree of the merged graph, two parts of candidate neighbors are combined. Any excess candidate neighbors are then truncated based on their distances to the vertex u, sorted in ascending order.

Algorithm 2 presents the detailed process of the *PGs to k-NNG transformation*. First, we initialize the algorithm and calculate the size of the search candidate set L based on *minimum cost querying* strategy (Lines 1-2). For each vertex u in each PG G_i, the algorithm acquires *local candidate neighbors* from its own PG and *cross candidate neighbors* from the other PGs (Lines 3-7). Then, the algorithm selects the top-k results from the combined candidate neighbors and assigns them to the vertex u in the merged k-NNG G (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 candidate set* strategy causes a critical challenge, that is: the strategy may lead to the *cross query* process easily fall

Algorithm 2: PGs to k-NNG Transformation

```
Input: set of the graph-based indexes $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$,
       maximum out-degree constraint $k$
Output: $k$-NNG $G$
1  Initialize the graph $G \leftarrow \emptyset$, candidate set $C \leftarrow \emptyset$
2  $L \leftarrow \frac{k}{|\mathcal{G}|-1}$
3  foreach $G_i \in \mathcal{G}$ do
4      foreach $\forall u \in V_i$ 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              $C \leftarrow C \cup KNNSearch(\vec{x}_u, G_j, L, L, ep_j)$
8          sort $C$ in ascending order of the distance to $\vec{x}_u$
9          resize $C$ to $k$
10         $N(u) \leftarrow C$
11 return $G$
```

into a local optimum, resulting in low-quality cross candidate neighbors in an inaccurate k-NNG. To address this issue, we propose an ***Indegree-Aware*** ***k-NNG Refinement*** approach to improve the quality of the merged graph-based index by iteratively refining the obtained k-NNG with the proposed *streamlined refinement* process while improving the graph connectivity via an *indegree augmentation* technique.

4.2.1 Streamlined Refinement. We refine the k-NNG through iterative NNDescent method [14]. In each iteration, the entire graph undergoes both *sampling* and *update* processes, where *sampling* refers to the process of acquiring neighbors for each vertex u in the graph, and *update* refers to the process of updating the neighbor set $N(u)$ for each vertex u based on the sampled neighbors. The complete algorithm of *sample* and *update* is provided in Appendix A of our technical report [45].

We observed that the results of *update* in each iteration directly affect the subsequent *sampling*. Notably, NNDescent performs batch *sampling* on the entire graph first, followed by another round of batch *updates*. To improve efficiency, we introduce a *streamlined refinement* approach, enabling *sampling* and *update* to be conducted simultaneously for each vertex. Specifically, we adopt an *update-before-sample* strategy. Given a k-NNG G, we initialize four auxiliary graphs to track samples: $G_{\mathtt{new}}$ and $G_{\mathtt{old}}$ store the *new* and *old* neighbors of each vertex, while $\overline{G}_{\mathtt{new}}$ and $\overline{G}_{\mathtt{old}}$ record their corresponding reverse neighbors (Line 1 in Algorithm 3), following the original method. For each vertex $v \in V$, the first iteration's *update* operation is based on the initial set of all neighbors in $N_G(v)$ (Lines 2-3). After completing the *update* process for v, its neighbors $N_G(v)$ are immediately sampled and added to the corresponding graph records (Lines 6-10). As each vertex is processed, the overall graph quality continuously improves, leading to progressively better *sampling* for subsequent vertices. With this approach, the k-NNG achieves faster convergence during the refinement process.

Figure 2(e) illustrates the k-NNG refinement. In this example, $\vec{x}_6$ is connected as the cross-graph neighbor of $\vec{x}_1$ in the pre-merged k-NNG, and subsequently serves as a pivot to facilitate connections among its neighbors by enabling them to recognize and establish

Algorithm 3: k-NNG Refinement

```
Input: $k$-NNG $G$, maximum iteration $I_{max}$
Output: Refined $k$-NNG $G$
1  Initialize the graph $G_{\mathtt{new}}, G_{\mathtt{old}}, \overline{G}_{\mathtt{new}}, \overline{G}_{\mathtt{old}} \leftarrow \emptyset$
2  foreach $u \in V$ do
3      Insert $\forall v \in N(u)$ into $G_{\mathtt{new}}[u]$
4  repeat
5      foreach $u \in V$ do
6          merge $\overline{G}_{\mathtt{new}}[u]$ into $G_{\mathtt{new}}[u]$, $\overline{G}_{\mathtt{old}}[u]$ into $G_{\mathtt{old}}[u]$
7          clear $\overline{G}_{\mathtt{new}}[u], \overline{G}_{\mathtt{old}}[u]$
8          perform Update for $u$
9          clear $G_{\mathtt{new}}[u], G_{\mathtt{old}}[u]$
10         perform Sample for $u$
11     $G \leftarrow RepairIndegree(G)$
12     $iter \leftarrow iter + 1$
13 until $iter = I_{max}$
14 return $G$
```

Algorithm 4: RepairIndegree(G)

```
Input: $k$-NNG $G$
Output: Repaired $k$-NNG $G$
1  $\mathcal{T} \leftarrow$ calculate in-degrees for $\forall u \in V$
2  foreach $\forall v^* \in V$ where $\mathcal{T}(v^*) = 0$ do
3      repeat
4          $v \leftarrow$ the closest unvisited neighbor in $N(v^*)$
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 $N(v)$
8                  Mark $v^*$ in $N(v)$ 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 $N(v^*)$ are visited
13 return $G$
```

direct links (e.g., linking $\vec{x}_3$ and $\vec{x}_4$). Since distant neighbors are replaced by closer neighbors, the quality of the candidate neighbors is improved.

4.2.2 Indegree Augmentation. Before augmenting, many vertices in G may have no in-edge, denoted as v^*. Given a k-NNG $G(V, E)$, this technique is designed to enhance the graph connectivity by linking disconnected vertices in G. Specifically, we try to connect each vertex v^* to its nearest neighbor $v \in N(v^*)$. To ensure that other vertices with low in-degrees 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 in-degree greater than 1 (Line 6 in Algorithm 4). If v' in $N(v)$ meets these conditions, we replace v' with v^* in $N(v)$, and mark v^* as *replaced* in $N(v)$ (Lines 7-11). Otherwise, we continue to visit the

Algorithm 5: k-NNG to PG Transformation

```
Input: $k$-NNG $G$, Indegree recorder $\mathcal{T}$, maximum
       out-degree constraint $k$
Output: Merged graph $\hat{G}$
1  Initialize the graph $\hat{G}(V', E') \leftarrow \emptyset$
2  foreach $\forall u \in V$ do
3      $\mathcal{R} \leftarrow \emptyset$
4      foreach $\forall v \in N(u)$ 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      $\hat{N}(u) \leftarrow \mathcal{R}$
9  foreach $\forall u \in V'$ do
10     $\hat{N}(u) \leftarrow \hat{N}(u) \cup \{v \in V' \mid (v,u) \in E'\}$
11     sort $\hat{N}(u)$ in ascending order of the distance to $\vec{x}_u$
12     resize $\hat{N}(u)$ to $k$
13 return $\hat{G}$
```

next closest neighbor $v \in N(v^*)$, following the ascending order of distance to $\vec{x}_{v^*}$, until no neighbors are left.

4.3 k-NNG to PG Transformation

Despite the refined k-NNG is 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 [24, 42], compared to state-of-the-art graph-based ANNS methods [16, 21, 28]. To bridge this gap, we propose a process to transform k-NNG back into PG, which consists of two main steps: 1) *neighbor selection* and 2) *connectivity enhancement*. 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 $u \in V$. Specifically, for each vertex u, we sort the neighbors in $N_G(u)$ in ascending order of the distance to $\vec{x}_u$, followed by acquiring up to k neighbors (Lines 2-8). The fundamental principle of this procedure is that a neighbor qualifies as a candidate neighbor if it is closer to u than to any other existing neighbors in the current candidate neighbor set. This approach is effective across commonly used distance metrics, such as ℓ_2 and cosine similarity, and can be extended to other metrics, including Maximum Inner Product Search (MIPS), by leveraging established transformations into ℓ_2 space, as demonstrated in recent studies [10]. To prevent any point in the graph from becoming unreachable, we enforce a constraint ensuring that pruned edges do not correspond to the only incoming connections of those points (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., MRNG [16] and τ-MNG [33]), allowing for flexibility in the choice of pruning methods.

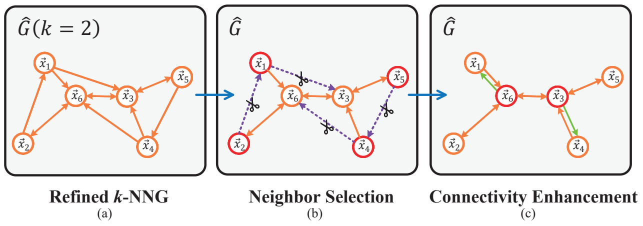


Figure 6: An example of *k-NNG to PG transformation.*

4.3.2 Connectivity Enhancement. To further improve the graph connectivity and avoid the degree budget waste, we conduct *connectivity enhancement* for the pruned graph $\hat{G}$. This process entails combining the reverse edges for each vertex $u \in V$, which is based on the principle that *people who value you highly are likely the ones you should value in return.* Specifically, we add reverse neighbors into neighbor set $N_{\hat{G}}(u)$ for each vertex u, and then sort the neighbors in $N_{\hat{G}}(u)$ in ascending order of the distance to $\vec{x}_u$ (Lines 8-10). Finally, the neighbor set $N_{\hat{G}}(u)$ is strictly truncated to k neighbors to avoid hubness (i.e., excessive out-degree) issues (Line 12). Notably, these additional edges facilitate query routing between distant vertices, thereby improving the graph's navigability for ANNS tasks.

As illustrated in Figure 6, the *neighbor selection* technique acquires candidate neighbors for each vertex based on a specific pruning strategy. For instance, in the candidate neighbor set of $\vec{x}_2$, we remove $\vec{x}_1$ since $\vec{x}_2$ is already connected to $\vec{x}_6$, and a shorter edge exists between $\vec{x}_1$ and $\vec{x}_6$. The *connectivity enhancement* process ensures the graph connectivity by adding reverse edges, as depicted by the green lines.

5 HIERARCHICAL INTEGRATION AND COMPLEXITY ANALYSIS

5.1 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 [28] method builds a multi-layer graph structure, where each layer contains vertices of varying density, thereby enabling superior search performance. Notably, the number of vertices in each layer of HNSW decreases exponentially with increasing layer depth, i.e., $P(level \geq k) \propto \exp(-k)$. 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 propose a *HNSW-Adaptive Merging Strategy*, where we first merge the base layer of HNSW using the proposed FGIM framework and subsequently reconstruct the higher layers accordingly.

To restore the hierarchical structure of HNSW, we propose a multi-level reconstruction algorithm, as shown in Algorithm 6. This algorithm begins by assigning a randomly determined level to each vertex in the base graph G and initializing the hierarchical graph-based index $\mathcal{G}$ with l_m levels, where l_m corresponds to the maximum level among all vertices (Lines 1-6). For each vertex $u \in V$ with a designated level $l > 1$, the algorithm greedily navigates to the closest vertex in the higher level $l+1$ and then searches for candidate neighbors C at each level i from l down to 1 using the *KNNSearch* function (Lines 7-14). Subsequently, the

Algorithm 6: Multi-level Reconstruction

```
Input: Merged PG $G$, out-degree constraint $m$, search
       parameter $L$
Output: Hierarchical graph-based index $\mathcal{G}$
1  $\mathcal{L} \leftarrow \{\}$
2  foreach $\forall u \in V$ do
3      $\mathcal{L}(u) \leftarrow$ generate random level
4  $l_m \leftarrow \max_{u \in V} \mathcal{L}(u)$
5  $ep \leftarrow$ randomly select one vertex from $l_m$ level
6  initialize $\mathcal{G}$ with $l_m$ levels and base graph $G$
7  foreach $\forall u \in V$ where $\mathcal{L}(u) > 0$ do
8      $l \leftarrow \mathcal{L}(u)$
9      $c \leftarrow ep$
10     foreach $i \leftarrow l_m$ to $l + 1$ do
11         $c \leftarrow$ KNNSearch($\vec{x}_u, \mathcal{G}(i), 1, 1, c$)
12     $C \leftarrow \emptyset$
13     foreach $i \leftarrow l$ to $1$ do
14         $C \leftarrow$ KNNSearch($\vec{x}_u, \mathcal{G}(i), L, L, c$)
15         $N_{\mathcal{G}(i)}(u) \leftarrow$ Heuristic($\vec{x}_u, C, m$)
16         foreach $\forall v \in N_{\mathcal{G}(i)}(u)$ do
17             $N_{\mathcal{G}(i)}(v) \leftarrow$ Heuristic($\vec{x}_u, N_{\mathcal{G}(i)}(v) \cup \{u\}, m$)
18         $c \leftarrow C[0]$
19 return $\mathcal{G}$
```

Heuristic function, introduced as *SelectNeighborsHeuristic* in the original paper [28], is applied to prune the candidate neighbors by the RNG pruning strategy [38] for u at each level, ensuring adherence to the out-degree constraint m (Line 15). Once candidate neighbors are identified, bidirectional edges are added between u and its neighbors. The same pruning process is systematically applied to all neighbors of u at each level (Lines 16-17).

5.2 Complexity Analysis

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

PGs to k-NNG transformation. The complexity of this transformation depends on the search complexity of the underlying graph-based index since our framework directly takes PG as input. Assuming that $\log\log n \ll d \ll \log n$ (i.e., the dataset's dimensionality is much smaller than its size), we consider two state-of-the-art graph-based ANNS methods: NSG [16] and HNSW [28], both of which have a search complexity of $O(\log n)$. Given h PGs, $\mathcal{G} = \{G_1, G_2, \ldots, G_h\}$, and an out-degree constraint k, the total complexity of this transformation is:

$$O\left(\frac{kd}{h-1}\sum_{i=1}^{h} n_i \sum_{j\neq i} \log n_j\right). \tag{5}$$

k-NNG refinement. Iterative update involves at most $I_{\max}$ iterations, where each iteration refines the graph using the *streamlined refinement* technique. The complexity of this step is $O(I_{\max} \cdot n \cdot k^2 d)$. The complexity of the *indegree augmentation* is $O(n \cdot k)$. Therefore, the total complexity of the *k-NNG refinement* is $O(I_{max} \cdot n \cdot k^2 d)$.

Table 2: The properties of the datasets.

Dataset	Dim	#Base	#Query	Metric	LID
SIFT [22]	128	1,000,000	10,000	ℓ_2	9.3
GIST [22]	960	1,000,000	1,000	ℓ_2	18.9
Deep [7]	96	1,000,000	10,000	cosine	12.1
Glove [34]	100	1,183,513	10,000	cosine	20.0
MSong [9]	420	994,185	1,000	ℓ_2	9.5
Crawl [1]	300	1,989,995	10,000	cosine	15.7

k-NNG to PG transformation. In this step, the complexity of the *neighbor selection* is $O(n \cdot k^2 d)$, and the complexity of the *connectivity enhancement* is $O(n \cdot k)$. Therefore, the total complexity of the *k-NNG to PG transformation* is $O(n \cdot k^2)$.

Summarizing all components, our FGIM framework consumes:

$$O\left(\frac{kd}{h-1}\sum_{i=1}^{h} n_i \sum_{j\neq i} \log n_j + I_{\max} \cdot n \cdot k^2 d\right) \tag{6}$$

time and $O(n \cdot k)$ space, where n_i is the number of vertices in graph G_i, k is the maximum out-degree constraint, h is the number of graph-based indexes, and $I_{\max}$ is the maximum number of iterations in the k-NNG refinement process.

6 EXPERIMENTAL STUDY

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

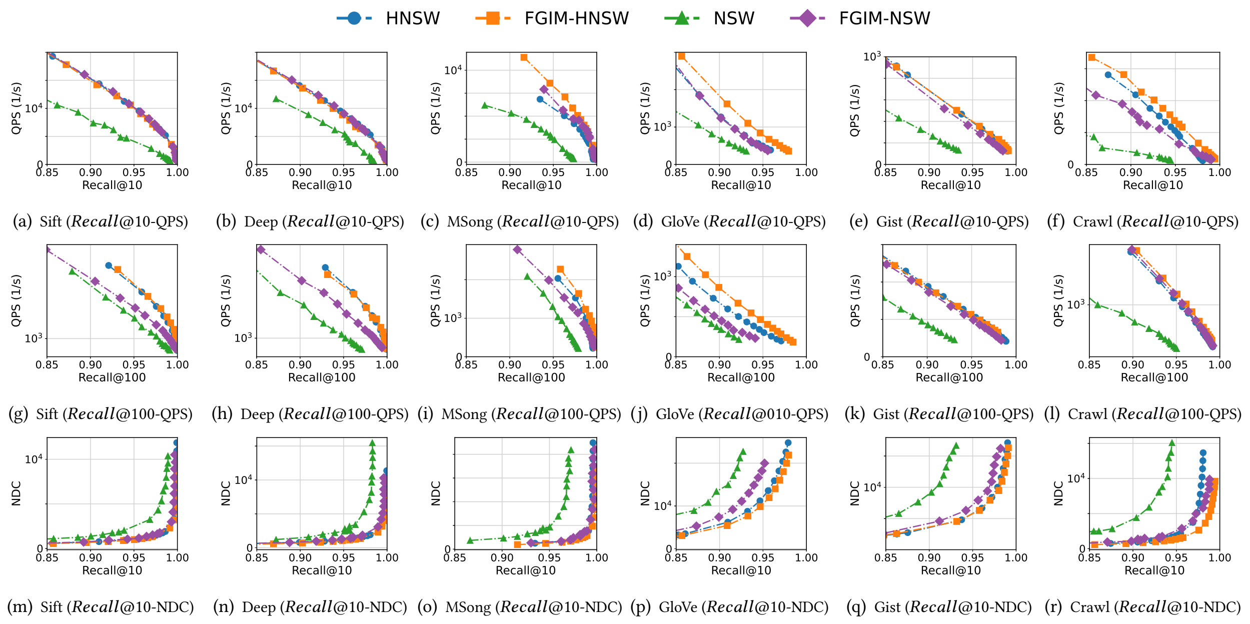
RQ1: How does FGIM compare to other methods in terms of merging efficiency and search performance? (**§6.2**)
RQ2: How does FGIM perform with other mainstream graph-based indexes? (**§6.3**)
RQ3: How does FGIM perform in merging multiple indexes? (**§6.4**)
RQ4: How do different strategies contribute to FGIM? (**§6.5**)
RQ5: How does FGIM scale with the dataset size? (**§6.6**)

We begin by introducing the experimental settings, followed by the main results to answer these above questions.

6.1 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. The datasets cover various applications such as image(SIFT [22], GIST [22], Deep [7]), text(Glove [34], Crawl [1]), and audio(MSong [9]). The properties of the datasets are summarized in Table 2. We also use the local intrinsic dimensionality (LID) [24, 42] to measure the difficulty of the datasets.

Compared algorithms. We consider applying our framework to merge several types of mainstream graph-based indexes to extend the generality of our framework. Specifically, we introduce several generic graph-based ANNS methods: (1) HNSW [28] is a state-of-the-art hierarchical graph-based index that is widely used in real-world systems. (2) Vamana [21] optimizes HNSW's neighbor selection strategy and is reported to achieve better search performance. (3) τ-MNG [33] is a recently proposed method that constructs a monotonic neighborhood graph based on an existing index. (4) NSW [27] is a representative navigable small-world graph.



(a) Sift (*Recall*@10-QPS) (b) Deep (*Recall*@10-QPS) (c) MSong (*Recall*@10-QPS) (d) GloVe (*Recall*@10-QPS) (e) Gist (*Recall*@10-QPS) (f) Crawl (*Recall*@10-QPS)

(g) Sift (*Recall*@100-QPS) (h) Deep (*Recall*@100-QPS) (i) MSong (*Recall*@100-QPS) (j) GloVe (*Recall*@010-QPS) (k) Gist (*Recall*@100-QPS) (l) Crawl (*Recall*@100-QPS)

(m) Sift (*Recall*@10-NDC) (n) Deep (*Recall*@10-NDC) (o) MSong (*Recall*@10-NDC) (p) GloVe (*Recall*@10-NDC) (q) Gist (*Recall*@10-NDC) (r) Crawl (*Recall*@10-NDC)

Figure 7: QPS, NDC vs. *Recall*@10 curves and QPS vs. *Recall*@100 curves for the merged index using FGIM and incremental Methods. (Exp.2)

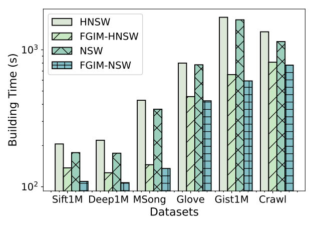


Figure 8: Merging efficiency. (Exp.1)

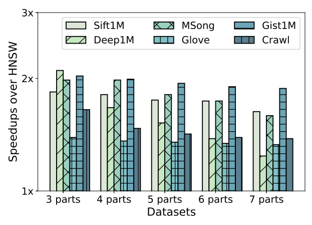


Figure 9: Merging efficiency in multiple indexes (Exp.6)

Table 3: Comparison of existing methods. (Time: minutes, *R*@10: *Recall*@10, Size: index size in MB) (Exp.3)

Dataset	NNMerge			DiskANN			FGIM (Ours)		
	Time	*R*@10	Size	Time	*R*@10	Size	Time	*R*@10	Size
Sift	3.24	0.961	220.5	17.66	0.995	352.6	2.78	0.996	123.3
Deep	3.23	0.943	220.6	19.04	0.992	372.8	3.28	0.994	134.3
MSong	6.31	0.963	219.3	29.42	0.996	270.5	5.43	0.997	92.3
GloVe	10.19	0.826	522.5	53.29	0.927	789.1	7.58	0.925	324.3
Gist	28.27	0.829	430.6	109.98	0.951	219.8	21.1	0.957	100.7
Crawl	32.04	0.863	925.6	120.42	0.987	955.6	20.9	0.978	277.2

(5) NNDescent [14] is a representative method for constructing a k-NNG. (6) NNMerge [47] is a recent merging method specifically designed for k-NNG. (7) DiskANN [21] proposes a merging strategy for indexing large-scale datasets.

Parameters. For each PG used in our experiments, we follow the approach of benchmarking papers [5, 24, 42] by employing grid search to determine the optimal parameter values, ensuring that the algorithm achieves its best search performance. For experiments requiring parameter variations, we will provide detailed discussions in the corresponding sections.

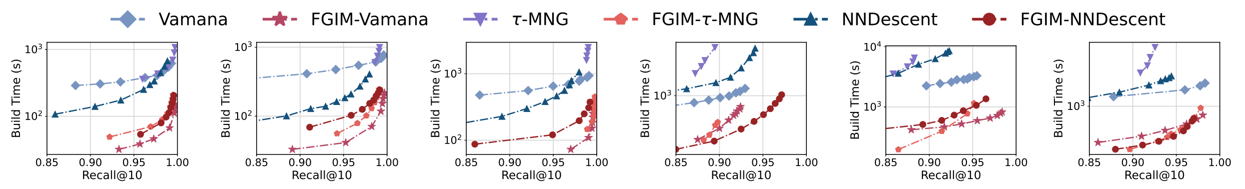
Computing Environment. We implemented the proposed method in C++11 and compiled the code using CMake 3.30.2 with GCC 11.4.0 as the compiler. For multi-threading, we utilized the OpenMP 4.5 library. All experiments were conducted on a machine equipped with an Intel Core i7-12700H CPU and 32 GB of RAM, running WSL2 Ubuntu 22.04 as the operating system. All results are averaged over five independent runs.

Measurements. To measure the accuracy of the search results, we use the *Recall* metric defined in §2.1. Additionally, Queries Per Second (QPS) and Number of Distance Computation (NDC) are used to measure the search efficiency. Generally, each method has its own parameters during a search. 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.

6.2 Efficiency Evaluation

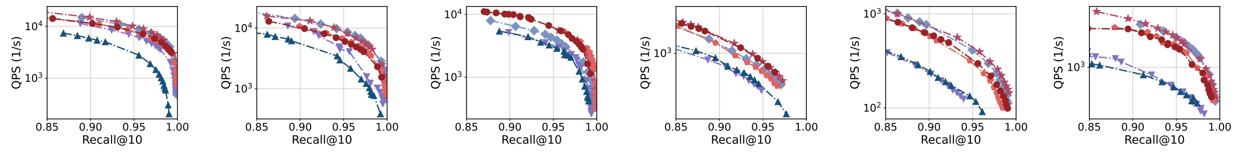
Exp.1 & 2: Comparison with Incremental Methods in Merging Efficiency and Search Performance. In this part, we study the merging efficiency and search performance of our FGIM framework in comparison to the incremental construction methods of HNSW and NSW. Each dataset is equally divided into two equal subsets, with two indexes constructed on each subset. For the baselines, we adopt their incremental approach, where one subset is sequentially inserted into the index built on the other. In contrast, our methods merge the two prebuilt indexes using the FGIM framework.

Figure 8 reports the construction cost of the merged index using the FGIM framework and the incremental method of HNSW and NSW. Our methods achieve substantially higher efficiency, demonstrating speedups of 1.49× and 1.62× on Sift, 1.72× and 1.64× on Deep, 2.95× and 2.71× on MSong, 1.75× and 1.83× on GloVe, 2.62×



(a) Sift (*Recall*@10-BT) (b) Deep (*Recall*@10-BT) (c) MSong (*Recall*@10-BT) (d) GloVe (*Recall*@10-BT) (e) Gist (*Recall*@10-BT) (f) Crawl (*Recall*@10-BT)

Figure 10: Construction time for our merging methods and the original methods rebuilt from scratch. (Exp.4)



(a) Sift (*Recall*@10-QPS) (b) Deep (*Recall*@10-QPS) (c) MSong (*Recall*@10-QPS) (d) GloVe (*Recall*@10-QPS) (e) Gist (*Recall*@10-QPS) (f) Crawl (*Recall*@10-QPS)

Figure 11: Search performance for our merging methods and the original methods rebuilt from scratch. (Exp.5)

and 2.79× on Gist, and 1.67× and 1.48× on Crawl, respectively, compared to the incremental approach. This improvement can be attributed to FGIM's ability to leverage both search mechanisms and iterative optimization strategies, thereby minimizing computational overhead. The results demonstrate that FGIM offers a more efficient approach than the conventional incremental method.

Figure 7 evaluates the search performance of incremental methods and our methods. Across all datasets, FGIM-based methods can achieve comparable or superior search performance compared to the incremental methods, demonstrating that our framework can effectively merge indexes while maintaining or enhancing search accuracy.

Exp.3: Comparison with other methods. As we discussed in §2.2, existing methods exhibit significant limitations. In this part, we evaluate these methods alongside our approach to demonstrate their shortcomings. Table 3 presents a comparison of different methods, focusing on merging time, *Recall*@10 with a fixed search parameter of 200, and index size. NNMerge achieves relatively fast merging but suffers from a lower *Recall*@10, consistent with findings in previous benchmark studies [24, 42], as k-NNG is not optimized for efficient ANNS. DiskANN's merging strategy, on the other hand, results in multiple indexing of the same point, significantly increasing index size and reducing efficiency. These results suggest that existing methods are incompetent for efficiently merging graph indexes.

In summary, our FGIM framework significantly outperforms other methods in terms of merging efficiency, achieving comparable or even superior search performance.

6.3 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.

Exp.4: Merging Efficiency. 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 their construction parameters to reconstruct indexes on the entire dataset from scratch, ensuring that each method has at least four different parameter configurations.

Figure 10 presents the experimental results. We can find that our methods fulfilled the merging task with high efficiency. Specifically, at the same Recall@10, (1) FGIM-Vamana achieved 3.4∼ 6.9× speedups over Vamana; (2) FGIM-τ-MNG achieved 4.3∼ 23.2× speedups over τ-MNG; (3) FGIM-NNDescent achieved 3.3∼ 6.7× speedups over NNDescent. On average, our method achieves a 7.4× speedup, demonstrating its efficiency in merging graph-based indexes over reconstructing an entirely new index from scratch.

Exp.5: Search Performance. As depicted in Figure 11, FGIM-Vamana, FGIM-τ-MNG, FGIM-NSW, FGIM-NNDescent consistently achieved comparable or superior performance compared to their original approaches. 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.

6.4 Multiple Indexes Merging

As discussed in §1, the ability to merge multiple graph-based indexes efficiently is essential for real-time systems that need to integrate frequently generated small indexes. To assess the effectiveness of our approach, we vary the number of indexes to be merged from 3 to 7, with each subset containing an equal number of vectors from the original dataset, and compare the merging cost against HNSW's incremental method.

Exp.6: Multiple 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.

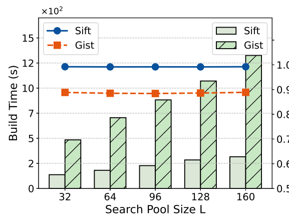


Figure 12: Effects of search pool size L (Exp.7)

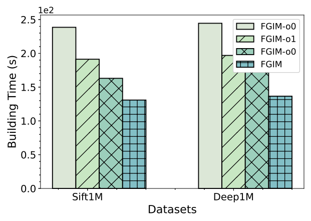


Figure 13: Effects of acceleration techniques (Exp.8)

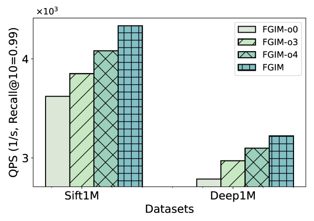


Figure 14: Effects of optimization techniques (Exp.9)

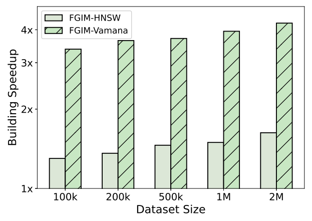


Figure 15: Scalability study (Exp.10)

6.5 Ablation Analysis

In this part, we assess the effects of our techniques used in the framework, i.e., the *Minimum Candidate Set Strategy*, the *cross-querying*, the *streamlined refinement*, the *indegree augmentation* and the reconstruction of the hierarchical graph structure.

Exp.7: Effects of Minimum Candidate Set Strategy. According to our strategy, when selecting cross-graph candidate neighbors, we set the search pool size 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 SIFT and 800 for GIST) and observe the changes in merging time and *Recall*@10, as shown in Figure 12. Our findings demonstrate that this strategy achieves the shortest index merging time without compromising graph quality, which can be explained that the iterative refinement allows the graph structure to converge to a similar structure.

Exp.8: Effects of Acceleration Technique. We first evaluate the effects of the *cross-querying* and the *streamlined refinement.* Notably, these techniques do not affect the final graph structure since the graph will converge to a similar optimal structure after iterative refinement, thus we only evaluate the merging efficiency. We define four variants for comparison: FGIM-o0 without any technique; FGIM-o1, which incorporates only *cross-querying* and adopts the original NNDescent for refinement; FGIM-o2, which employs only *streamlined refinement* and initializes the merged index by randomly introducing vertices from the other indexes. We report the building time of the merged index on the Sift dataset in Figure 13. Overall, each technique contributes to a significant acceleration of the merging process. Since these optimizations are mutually independent, their combined application FG yields the lowest overall construction cost.

Exp.9: Effects of Performance Optimization. Figure 14 evaluates the effects of the *indegree augmentation* and the reconstruction of hierarchical graph structure. Following the denotation in Exp.6, we define FGIM-o3 as the method incorporating only hierarchical graph reconstruction and FGIM-o4 as the method utilizing only *indegree augmentation.* The results indicate that both techniques contribute to improved search performance, with FGIM-o4 yielding a more significant enhancement than FGIM-o3. This can be attributed to the effectiveness of *indegree augmentation* in enhancing graph connectivity, thereby facilitating navigation to disconnected vertices.

6.6 Scalability Study

Exp.10: Scalability Study. Figure 15 reports an evaluation of the scalability of our framework on the Sift dataset. 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.

7 RELATED WORK

Recently, with the rise of *Large Language Models* (LLMs) and *retrieval-augmented generation* (RAG), ANNS 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 [5, 31] have shown that graph-based methods [14, 16, 28] outperform traditional methods such as hashing-based methods [19, 20, 37], tree-based methods [8, 30, 36], inverted index-based methods [6, 7, 46], and quantization-based methods [17, 22] in terms of search quality and efficiency.

The mainstream graph-based methods [15, 16, 21, 27, 28, 33] build a proximity graph where each node is a base vector and edges connect several nearby vectors. Among them, HNSW [28] is a widely used graph-based method that is derived from small-world networks and is constructed incrementally in an online fashion. Other state-of-the-art methods, such as NSG [16] and τ-MNG [33], construct indexes by performing refinement operations on the existing graph (e.g., k-NNG). Due to their superior performance, these methods have been widely adopted in production [40, 43]. Our proposed framework is compatible with the methods mentioned above and can be adapted to merge their indexes.

8 CONCLUSION

In this paper, we study the problem of merging existing graph-based indexes into a single one for effective ANNS. To achieve this goal, we propose a general FGIM framework with three core techniques. First, we introduce a *PGs to k-NNG transformation* that consists of *cross-querying* and *top-k selection* to extract candidate neighbors from the existing graph-based indexes. Second, we propose a streamlined and indegree-aware *k-NNG refinement* method to improve candidate neighbors' quality. Finally, the *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 state-of-the-art methods without compromising the search performance.

REFERENCES

[1] Anon. Retrieved April 15, 2020. Common Crawl. http://commoncrawl.org/.

[2] Sunil Arya and David M Mount. 1993. Approximate nearest neighbor queries in fixed dimensions.. In *SODA*, Vol. 93. Citeseer, 271–280.

[3] 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.

[4] 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.

[5] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.

[6] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.

[7] 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.

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

[9] 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)*.

[10] 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).

[11] Scott Cost and Steven Salzberg. 1993. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine learning* 10 (1993), 57–78.

[12] 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.

[13] 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).

[14] 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.

[15] 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.

[16] 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).

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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).

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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).

[32] 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.

[33] 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.

[34] 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.

[35] Manan D Shah and Harshad B Prajapati. 2013. Reallocation and allocation of virtual machines in cloud computing. *arXiv preprint arXiv:1304.3978* (2013).

[36] 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.

[37] 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).

[38] Godfried T Toussaint. 1980. The relative neighbourhood graph of a finite planar set. *Pattern recognition* 12, 4 (1980), 261–268.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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

[44] Shuo Yang, Jiadong Xie, Yingfan Liu, Jeffrey Xu Yu, Xiyue Gao, Qianru Wang, Yanguo Peng, and Jiangtao Cui. 2024. Revisiting the Index Construction of Proximity Graph-Based Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2410.01231* (2024).

[45] Peng Cheng Xiaoyao Zhong Lei Chen Zhitao She Jingkuan Song Xiaofeng Cao Heng Tao Shen Xuemin Lin Zekai Wu, Jiabao Jin. 2025. Fast Graph-based Indexes Merging for Approximate Nearest Neighbor Search [technical report]. http://cspcheng.github.io/pdf/FastMerging.pdf. (2025).

[46] Peitian Zhang and Zheng Liu. 2022. Bi-Phase Enhanced IVFPQ for Time-Efficient Ad-hoc Retrieval. *arXiv preprint arXiv:2210.05521* (2022).

[47] 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.

[48] 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.

A DETAILS OF k-NNG REFINEMENT

In this section, we present the details of the k-NNG refinement component [14], i.e., *update* function and *sample* function.

The *update* function iterates over all pairs of vertices $\langle v_1, v_2 \rangle$ within $G_{\mathsf{new}}[u]$ and between $G_{\mathsf{new}}[u]$ and $G_{\mathtt{old}}[u]$ (Line 1). For each pair, it computes the distance $\mathcal{D}$ between v_1 and v_2 and extracts the farthest neighbors in $N(v_1)$ and $N(v_2)$ (Line 2). If $\mathcal{D}$ is smaller than the distance to the farthest neighbor, the algorithm replaces the farthest neighbor with the new vertex (Lines 3-6). This iterative process continues until all pairs have been evaluated, facilitating mutual recognition between neighboring vertices in $N(u)$ and establishing direct connections between them.

The *sample* function is responsible for categorizing the neighbors of vertex u into the recorders $G_{\mathtt{old}}$, G_{new}, $\overline{G}_{\mathtt{old}}$, and $\overline{G}_{\mathsf{new}}$. It first determines whether a neighbor v is new or old based on its flag (Line 2). If v is new, it is added to $G_{\mathsf{new}}[u]$, and u is added to $\overline{G}_{\mathsf{new}}[v]$ (Lines 3-5), after which the vertex is marked as old (Line 6). Otherwise, v is inserted into $G_{\mathtt{old}}[u]$, and u is added to $\overline{G}_{\mathtt{old}}[v]$ (Lines 7-8). This flag-based approach can helpfully reduce redundant computations, without changing the final results.

Algorithm 7: Update

```
  Input: $k$-NNG $G$, Graph $G_{\mathtt{old}}$, $G_{\mathsf{new}}$, vertex $u$
  Output: Refined $k$-NNG $G$
1 foreach $v_1, v_2 \in G_{\mathsf{new}}[u]$, $v_1 < v_2$ or $v_1 \in G_{\mathsf{new}}[u]$, $v_2 \in G_{\mathtt{old}}[u]$ do
2     $\mathcal{D} \leftarrow \delta(v_1, v_2)$
3     $v_1' \leftarrow$ the farthest neighbor in $N(v_1)$
4     replace $v_1'$ with $\langle v_2, d, \mathsf{new} \rangle$ if $\mathcal{D} < \delta(v_1, v_1')$
5     $v_2' \leftarrow$ the farthest neighbor in $N(v_2)$
6     replace $v_2'$ with $\langle v_1, d, \mathsf{new} \rangle$ if $\mathcal{D} < \delta(v_2, v_2')$
7 return $G$
```

Algorithm 8: Sample

```
  Input: Neighbor set $N(u)$, Graph $G_{\mathtt{old}}$, $G_{\mathsf{new}}$, $\overline{G}_{\mathtt{old}}$, $\overline{G}_{\mathsf{new}}$,
         vertex $u$
  Output: $G_{\mathtt{old}}$, $G_{\mathsf{new}}$, $\overline{G}_{\mathtt{old}}$, $\overline{G}_{\mathsf{new}}$
1 foreach $\forall v \in N(u)$ do
2     if $v$ is new then
3         insert $v$ into $G_{\mathsf{new}}[u]$
4         insert $u$ into $\overline{G}_{\mathsf{new}}[v]$
5         mark $v$ as old
6     else
7         insert $v$ into $G_{\mathtt{old}}[u]$
8         insert $u$ into $\overline{G}_{\mathtt{old}}[v]$
9 return $G_{\mathtt{old}}$, $G_{\mathsf{new}}$, $\overline{G}_{\mathtt{old}}$, $\overline{G}_{\mathsf{new}}$
```