# ESG: Elastic Graphs for Range-Filtering Approximate $k$-Nearest Neighbor Search

Mingyu Yang[1,2], Wentao Li[3], Zhitao Shen[4], Chuan Xiao[5], Wei Wang[1,2]
The Hong Kong University of Science and Technology (Guangzhou), China[1]
The Hong Kong University of Science and Technology, China[2]
University of Leicester, UK[3],Ant Group, China[4],Osaka University, Japan[5]
myang250@connect.hkust-gz.edu.cn,wl226@leicester.ac.uk
zhitao.szt@antgroup.com,chuanx@ist.osaka-u.ac.jp,weiwcs@ust.hk

## Abstract

Range-filtering approximate $k$-nearest neighbor (RFAKNN) search takes as input a vector and a numeric value, returning $k$ points from a database of $N$ high-dimensional points. The returned points must satisfy two criteria: their numeric values must lie within the specified query range, and they must be approximately the $k$ nearest points to the query vector. A straightforward approach to processing RFAKNN queries is to filter out points outside the query range and then perform a search on the remaining in-range points. While indexing can speed up this process, ensuring correctness would require indexing all possible $O(N^2)$ query ranges, which is infeasible due to severe storage constraints. Lossy compression techniques have been explored to create compact indexes for these ranges, but they often compromise query accuracy because of the inherent loss of information. To address these challenges, existing methods index only a subset of ranges, organized within a segment tree. At query time, the query range is decomposed into multiple subranges stored in the segment tree. Although these methods improve accuracy, they introduce significant query overhead since processing a single query requires combining indexes from $O(\log N)$ subranges. To strike a better balance between query accuracy and efficiency, we propose novel methods that relax the strict requirement for subranges to *exactly* match the query range. This elastic relaxation is based on a theoretical insight: allowing the controlled inclusion of out-of-range points during the search does not compromise the bounded complexity of the search process. Building on this insight, we prove that our methods reduce the number of required subranges to at most *two*, eliminating the $O(\log N)$ query overhead inherent in existing methods. Extensive experiments on real-world datasets demonstrate that our proposed methods outperform state-of-the-art approaches, achieving performance improvements of 1.5x to 6x while maintaining high accuracy.

## 1 Introduction

The problem of $k$-nearest neighbor (KNN) search in high-dimensional spaces has received increasing attention, especially with recent advancements in deep learning and large language models [27]. Due to the curse of dimensionality [23], finding the exact $k$ nearest neighbors often becomes computationally demanding. As a result, the variant known as **approximate $k$-nearest neighbor (AKNN)** search has been extensively studied. Numerous methods have been proposed for AKNN search [1, 3–5, 8, 12, 14–16, 18, 19, 22, 25, 26, 36, 38, 43, 48, 51], among which graph-based approaches [10, 11, 21, 24, 29, 35] have emerged as particularly effective. Graph-based methods, such as HNSW [31], build graphs as the index for AKNN search and demonstrate superior performance compared to alternative methods.

Driven by the demands of applications, **Range-Filtering Approximate $k$-Nearest Neighbors (RFAKNN)** search extends the traditional AKNN search problem by incorporating additional numerical constraints. In such scenarios, data points typically consist of two components: a vector and an associated numerical attribute. Formally, given a database $\mathcal{D}$ of $N$ data points, where each point $v_i$ is associated with a numerical value[1] $i$, an RFAKNN query takes as input a vector $q$ and a range $[l_q, r_q]$, and returns $k$ points as the result. These points should satisfy two conditions: their numerical values must lie within the query range $[l_q, r_q]$, and they must be approximately the $k$ closest points to the query vector $q$. By introducing a range as a filter to exclude data points outside the specified range, RFAKNN enables more targeted and meaningful nearest-neighbor retrieval. For example, in e-commerce applications, each product typically has a price attribute. By specifying a price range during an AKNN search, users can filter out products beyond their budget, improving the relevance of search results. Similarly, RFAKNN has broad applicability in Retrieval-Augmented Generation (RAG) [9], vehicle search [52], and face recognition [44].

**Search Principles.** To process RFAKNN queries, adopting a graph index (such as HNSW) originally designed for AKNN search is a straightforward approach. The main challenge lies in handling the additional query ranges and two commonly used principles for this are: 1) PreFiltering, which filters out points whose numeric values fall outside the query range. Since it ensures that only in-range points are included, the resulting graph index may become less-connected after removing out-of-range points. Conducting RFAKNN searches on such a sparse graph often leads to reduced query accuracy. 2) PostFiltering, which retains all points, including those that fall outside the query range. It progressively identifies points near the query point but only adds a point to the result if its numeric values fall within the query range. Although PostFiltering avoids the sparsity issue, it can operate on a graph index containing a large number of out-of-range points, which can slow down query processing. While further optimizations such as SuperPostFiltering [9] have been proposed to improve query speed, these often come at the cost of increased space requirements.

**State-of-the-art.** The aforementioned methods typically assume the creation of a *single* graph index for all points in $\mathcal{D}$, which is then adapted for RFAKNN searches using either the PreFiltering or

[1]We apply the re-ranking strategy in [52] to map the attribute value of a point $v_i$ to its position $i$ in $\mathcal{D}$. Thus, both the cardinality of $\mathcal{D}$ and the range are $N = |\mathcal{D}|$.

**Table 1: We compare our methods with existing solutions. Among them, PreFiltering, PostFiltering, and SuperPostFiltering primarily focus on search within a single graph index. SeRF[52] is a compression-based method, while SegmentTree[9] and IRANGE [45] are reconstruction-based approaches. The Scalability and Query Efficiency are confirmed by our experimental study. Theoretical Guarantee indicates whether the methods provide query time complexity analysis. Adaptability assesses if the methods can extend beyond graph-based indexing (for each range). Additionally, Half-Bounded RFAKNN is to check whether tailored methods are designed specifically for this type of query.**

| Feature | ESG (Our) | PreFiltering | PostFiltering | SuperPostFiltering [9] | SeRF [52] | SegmentTree [9] | IRANGE [45] |
|---|---|---|---|---|---|---|---|
| Scalability | ★★★ | ★★★ | ★★★ | ★★ | ★★★ | ★★★ | ★★ |
| Query Efficiency | ★★★ | ★ | ★ | ★★★ | ★★ | ★★ | ★★★ |
| Theoretical Guarantee | ✓ | × | × | × | × | ✓ | × |
| Adaptation | ✓ | ✓ | ✓ | ✓ | × | ✓ | × |
| Half-Bounded RFAKNN | ✓ | × | × | × | ✓ | × | × |

PostFiltering principles. To address the limitations of these methods, more advanced techniques have been developed that focus on creating *multiple* graph indexes. Recall that, in RFAKNN, users can specify up to $O(N^2)$ possible query ranges $[l_q, r_q]$, as both $l_q$ and $r_q$ are bounded by the cardinality $N$ of the database. If we create the graph index for each possible range, then when a query with range $[l_q, r_q]$ is issued, the precomputed graph index corresponding to range $[l_q, r_q]$ is utilized for query processing, which ensures both query accuracy and efficiency. Yet, maintaining $O(N^2)$ graph indexes demands an exhaustive amount of storage.

To avoid the need for materializing $O(N^2)$ graph indexes, existing methods aim to reduce the amount of stored information, and can be categorized into two types: 1) Compression-based methods [52], which exploit the relationships between graph indexes of different ranges to enable compression, which achieves an average index size of $O(N \log N)$. However, these methods employ lossy compression. As a result, if the index information for a specific query range is not encoded in the compressed representation, query accuracy may degrade. 2) Reconstruction-based methods [9, 45], which select a subset of ranges from the total $O(N^2)$ possible query ranges to construct graph indexes, and these selected ranges are organized as a segment tree. When an RFAKNN query with a range $[l_q, r_q]$ is received, the range $[l_q, r_q]$ is decomposed into subranges, which are precisely recorded in the segment tree. The graph indexes corresponding to these subranges are then utilized to reconstruct the final graph index for the query over the range $[l_q, r_q]$. The segment tree plays a crucial role in recovering unrecorded ranges, ensuring the accuracy of the RFAKNN search query. However, this accuracy comes with a cost. To reconstruct the range for a query, the process will decompose the range into as many as $O(\log N)$ subranges, which negatively impacts query efficiency.

**Theoretical Findings.** An intriguing observation is that both compression-based methods and current reconstruction-based approaches adhere to the PreFiltering principle: they construct the graph index only after excluding *all* out-of-range points. However, employing PreFiltering introduces drawbacks: compression-based methods fail to guarantee query accuracy, and reconstruction-based approaches resolve this limitation but at the expense of reduced query efficiency. This raises a critical question: is it possible to achieve both query accuracy and efficiency simultaneously? Surprisingly, our findings reveal that adopting the PostFiltering principle provides a promising solution to this question. PostFiltering ensures query accuracy by including all data points during the search. Yet, this accuracy comes at the cost of increased query overhead due to the inclusion of *all* out-of-range points in $\mathcal{D}$, making a direct application of PostFiltering *inefficient*. Recall that PostFiltering operates over the entire range $[1, N]$. To address this, we investigate the behavior of the graph index when applied to any superset range $[a, b]$ of the query range $[l_q, r_q]$, where $l_q \geq a$ and $r_q \leq b$, rather than only on range $[1, N]$ used in PostFiltering. In this case, the graph index for the superset $[a, b]$ includes all in-range points as well as *some* out-of-range points for query range $[l_q, r_q]$.

Our first theoretical finding is that, for a query range $[l_q, r_q]$, using a graph index corresponding to a superset range $[a, b]$ can correctly answer the RFAKNN query for $[l_q, r_q]$. This also explains why PostFiltering guarantees query accuracy (but over-killed): it builds the index over the entire range $[1, N]$, which is the superset of all possible query ranges. Our second theoretical finding is that, if the size of the superset range $[a, b]$ is only $\beta$ times larger than the query range $[l_q, r_q]$, the query time complexity increases by merely a linear factor compared to the original search time. This insight paves the way for a more efficient alternative: Instead of using the graph index for the full range $[1, N]$ as required by PostFiltering, we propose using a graph index corresponding to a tight superset range $[a, b]$ that is only slightly larger than the range $[l_q, r_q]$. In this way, we preserve the accuracy guarantees of PostFiltering while improving query efficiency.

**Our Novel Methods.** When combining the aforementioned theoretical findings with current reconstruction-based methods, we conceived the idea for our novel design: Elastic Graph (ESG), a new method for RFAKNN search. We first address the problem of answering half-bounded RFAKNN queries, where the input range has the form $[1, r_q]$, with $r_q \in [1, N]$ being the only flexible variable. We show that, by creating graph indexes for only $\log N$ ranges of the form $[1, N/2^i]$, where $i \in [0, \log N]$, RFAKNN queries with range $[l_q, r_q]$ can be efficiently resolved by referencing its tightest superset. We also prove that the selected superset range is at most twice the size of the query range, ensuring query efficiency. Also, instead of constructing the graph indexes for these $\log N$ ranges $[1, N/2^i]$ individually, we propose a holistic indexing approach that constructs a single range that generates the index for all necessary subranges, streamlining the indexing process.

Next, we extend our method to support general RFAKNN queries, where the query range $[l_q, r_q]$ includes two flexible variables, $l_q$ and $r_q$. Our approach also leverages a segment tree, with a key innovation in query processing: instead of identifying subranges that *exactly* match the query range, we allow the combined results of subranges to form a superset of the input range. We theoretically demonstrate that this elastic partitioning ensures the query range can be divided into at most *two* subranges, thereby eliminating

the $O(\log N)$ factor present in the query processing of existing reconstruction-based solutions. Furthermore, we propose a novel method for constructing graph indexes for all ranges in the segment tree, significantly accelerating index construction. Table 1 presents a comparison between our proposed method and the existing approaches for RFAKNN queries.

**Contributions.** We summarize our contributions as follows:

*Problem Analysis of State-of-the-art (§ 2).* We classify existing methods for answering RFAKNN queries into two main categories. We find that compression-based methods often compromise on query accuracy, whereas reconstruction-based methods suffer from query inefficiencies. This motivates our investigation into novel approaches for processing RFAKNN queries.

*Theoretical Findings (§ 3.)* To balance query accuracy and efficiency, we leverage the PostFiltering principle and investigate the relationship between a graph index constructed on a superset of a range (to accommodate out-of-range points) and the original range. Our findings demonstrate that a graph index built on the superset range ensures accurate query processing when the original range is queried. Moreover, we establish that the query time for the graph constructed on the superset range is bounded by a constant factor of the query time for the original range. These results extend the existing PostFiltering principle, laying the groundwork for the development of our novel methods.

*Novel RFAKNN Query Processing Methods (§ 4).* We first propose a novel index structure tailored for the half-bounded RFAKNN search, based on our theoretical findings. This index is constructed over $\log N$ ranges and is proven to be sufficient for answering range queries for all input ranges. We then design a new index structure for the general RFAKNN search, leveraging a segment tree approach. Our main technical contribution lies in introducing a novel query-processing method. This method ensures that the query range is divided into at most two subranges in the worst case.

*Extensive Experimental Studies (§ 5).* We evaluated our proposed methods on several real-world datasets and compared them with state-of-the-art methods. The experimental results demonstrate that our approaches outperform existing methods in both query efficiency and accuracy. Moreover, our methods are scalable to datasets with up to 100 million data points, making it desirable for large-scale RFAKNN search.

## 2 Preliminary

We begin by introducing the problem of approximate $k$-nearest neighbors (AKNN) search and its graph-based solutions. Following this, we formally define the range-filtering AKNN search problem and provide an overview of existing approaches. For clarity, the commonly used notations are summarized in Table 2.

### 2.1 AKNN Search and Graph-Based Solutions

The $k$-nearest neighbors (KNN) search in high-dimensional spaces involves retrieving the top $k$ closest vectors to a given query point based on a specified distance metric. This study takes the Euclidean distance as an instance, as the methods used for handling Euclidean distance can be extended to other distance metrics.

**Table 2: A Summary of Notations**

| Notation | Description |
|---|---|
| $\mathcal{D}$ | A set of vectors/points with numerical attributes |
| $N$ | The cardinality of $\mathcal{D}$ |
| $q$ | The query vector |
| $e$ | The elastic factor |
| $d$ | The dimensionality of $\mathcal{D}$ |
| $\mathcal{N}(u)$ | The neighbors of node $u$ in a graph index |
| $[l, r]$ | The range with left bound $l$ and right bound $r$ |
| $\|[l, r]\|$ | The size of range $[l, r]$ |
| $R$ | The set of ranges |
| $\mathbb{R}^d$ | The $d$-dimensional Euclidean space |
| $\|u, v\|$ | The Euclidean distance between $u$ and $v$ |
| $\ell$ | The range length bound |
| $\mathcal{I}$ | The graph index such as HNSW |
| $\text{ESG}_{1D}$, $\text{ESG}_{2D}$ | The elastic graph indexes |

*Definition 2.1.* Let $\mathcal{D} = \{v_1, v_2, \ldots, v_N\}$ be a set of $N$ points/vectors in $\mathbb{R}^d$. Given a query point $q \in \mathbb{R}^d$ and an integer $k > 0$, the **k-nearest neighbors (KNN) query** returns the set of $k$ points $S \subseteq \mathcal{D}$ with the smallest Euclidean distances to $q$. Formally, $\forall v \in \mathcal{D} \setminus S$, $||q - u|| \leq ||q - v||$ for all $u \in S$.

Due to the curse of dimensionality [23], searching for the exact KNN of a query point in high-dimensional space is computationally expensive. Therefore, a relaxed version of the KNN search, known as the **Approximate K-Nearest Neighbors (AKNN)** search, has been proposed to return the Approximate KNN for a query point.

**Graph Index.** To process AKNN queries, numerous methods have been proposed. Among these, graph-based algorithms [10, 11, 21, 24, 29, 35] have gained great attention due to their state-of-the-art search performance. These algorithms construct a graph $G$ as an index, where all points in $\mathcal{D}$ are represented as vertices, and edges are carefully selected to satisfy the monotonic search network (MSNET) property [11]. Specifically, the graph index leverages an edge occlusion strategy [11, 21, 24, 31, 34, 35] to prune redundant edges while maintaining the essential MSNET properties, which underpin the effectiveness of these methods. Additionally, theoretical studies demonstrate that the degree of graph $G$ after edge occlusion remains bounded by a constant, ensuring both efficiency and scalability.

**Search Algorithm.** When the graph index $G$ (e.g., HNSW [31]) is created, Algorithm 1 shows the process of searching for AKNN of a query point $q$. Initially, the entry point $ep$ is inserted into two heaps, $P$ and $Q$, where $P$ is used to set priorities and $Q$ is used to store the current top-$m$ closest neighbors (Line 1–2), where $m \geq k$ is the beam search size. Algorithm 1 proceeds by repeatedly popping a point $u$ from $P$ until $P$ becomes empty (Line 3–4). For each point $u$, its distance to the query point $q$ is compared with the largest distance between $q$ and the points in $Q$. If $u$'s distance is greater than this largest distance, the search stops (Line 5). Otherwise, for each neighbor $v \in \mathcal{N}(u)$ of $u$ in $G$ (Line 6), it checks whether $v$ has already been visited (Line 7). If visited, it is skipped; otherwise $v$ is inserted into both $P$ and $Q$ (Line 9, 11). When the size of the answer

**Algorithm 1:** Graph-Search$(G, ep, q, m)$

```
1  P ← ep ;          // priority queue, min heap by distance
2  Q ← ep ;            // result queue, max heap by distance
3  while P ≠ ∅ do
4      u ← P.top();
5      if ||q, u|| > ||q, Q.top()|| then break;
6      for v ∈ N(u) do
7          if v is visited then continue;
8          //PreFiltering: If v is out-of-range then Continue;
9          P ← P ∪ v;
10         //PostFiltering: If v is out-of-range then Continue;
11         Q ← Q ∪ v;
12     if Q.size() > m then
13         Q.resize(m)// keep top m results
14 return Q;
```

queue $Q$ exceeds $m$, the algorithm adjusts by removing points with larger distances to $q$ to maintain the desired size (Line 12–13). The top-$k$ closest points in $Q$ are then returned as the answer (Line 14).

## 2.2 RFAKNN Search and Existing Solutions

When a point has an associated numeric attribute (e.g., price), the problem of the AKNN search can be more flexible. This flexibility arises because we can pre-assign a range filter in the query to eliminate points whose numeric values do not fall within the specified range. In light of this, we define the **Range-Filtering AKNN (RFAKNN)** search problem as follows.

*Definition 2.2.* Let $\mathcal{D} = \{v_1, v_2, \ldots, v_N\}$ be a set of $N$ points in $\mathbb{R}^d$, where each point $v_i$ is associated with a numerical value $i \in [1, N]$. The **RFAKNN query** is defined as $Q = (q, l, r)$, where $q$ is a vector and $[l, r]$ denotes a range. The RFAKNN query returns the result of AKNN search of $q$ in $\mathcal{D}_{[l,r]}$, where $\forall v_i \in \mathcal{D}_{[l,r]}$, $i \in [l, r]$.

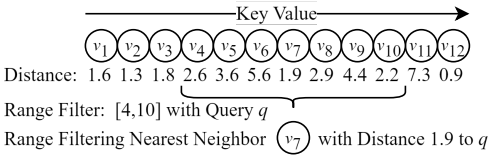



**Figure 1: The example of the RFAKNN query, where the points $v_i$ has an additional numerical attribute $i$. Given a query point $q$ with the range $[4, 10]$, the answer (for $k = 1$) is $v_7$ because the distance from $q$ to $v_7$ is the smallest among all in-range points $\mathcal{D}_{[4,10]} = \{v_4, v_5, \cdots, v_{10}\}$.**

PreFiltering **and** PostFiltering **Principles.** There are two principles for answering RFAKNN search queries. Both methods build upon the search algorithm presented in Algorithm 1, which was originally designed for AKNN queries.

• PreFiltering: Using this principle, we identify and discard out-of-range points during the search (Line 9 of Algorithm 1). Specifically, if the numeric value of a point $v$ is not within the query range $[l, r]$, then $v$ is excluded from the search process (i.e., it is not added to the queue $P$). In this way, the RFAKNN query under range $[l, r]$ can be answered by ensuring that only in-range points are considered.

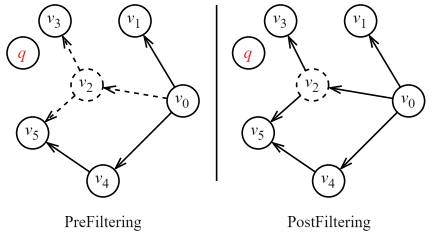



**Figure 2: The Illustration of** PreFiltering **and** PostFiltering

• PostFiltering: This principle does not discard out-of-range points during the search phase. Instead, only when a point is encountered and evaluated for inclusion in the result queue $Q$, it discards the point if its numeric value is not within the query range $[l, r]$ (Line 10 of Algorithm 1).

Example 1. *In Fig. 2, let $v_0$ be the entry node and $q$ be the query point. Assume $v_2$ is an out-of-range point for a given RFAKNN query. According to the* PreFiltering *principle, all edges connected to $v_2$ (depicted as dashed lines) are removed from the graph index $G$. This removal results in the disconnection between $v_0$ and the nearest neighbor of $q$, $v_3$. Thus, the search algorithm returns $v_5$, traversing the path $v_0 \to v_4 \to v_5$. In contrast, the* PostFiltering *principle retains the edges connected to $v_2$ for searching purposes, while excluding $v_2$ itself as a candidate result. Thus, it successfully identifies the nearest neighbor, $v_3$, by traversing the path $v_0 \to v_2 \to v_3$, leveraging the outgoing edges of $v_2$.*

*Drawbacks.* The PreFiltering principle is effective in removing out-of-range points from the graph, which reduces the search space. As a result, PreFiltering ensures high query efficiency. However, since PreFiltering removes some necessary points from the graph, it may cause the graph to become sparse, potentially preventing the search from finding the correct result. In contrast, PostFiltering directly searches on the whole graph, maintaining its high accuracy. Nevertheless, it requires computing the distances of many out-of-range points, which can result in a longer query time.

**State-of-the-art.** Recall that PreFiltering and PostFiltering only work on a *single* graph index created for the entire range $[1, N]$ (i.e., all points in the whole database $\mathcal{D}$). To overcome their limitations, existing solutions primarily focus on creating *multiple* graph indexes, and these solutions can be categorized into two types.

• Compression-based methods. These methods conceptually construct the graph index for each of the $O(N^2)$ possible ranges. We denote that a graph index is created for a range $[l, r]$ when the graph is actually created for points in $\mathcal{D}_{[l,r]}$, where $l, r \in [1, N]$. When an RFAKNN query is issued, the graph corresponding to the query range $[l_q, r_q]$ is retrieved, and the query is processed using a graph search algorithm (e.g., Algorithm 1). Since materializing and storing all $O(N^2)$ possible ranges is infeasible, these methods propose a holistic index that compactly compresses these $O(N^2)$ indexes to a size of $O(N \log N)$. However, current compression-based methods are lossy, meaning that the information corresponding to a range $[l, r]$ may be incomplete. As a result, queries for such ranges may lack accuracy. For instance, the RFAKNN query performance deteriorates if the length of a query range is small, making it difficult to achieve a recall of 0.8 or higher.

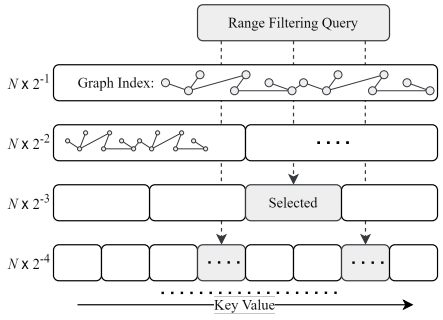



**Figure 3: The Example of Reconstruction-based Methods for the RFAKNN Query**

• Reconstruction-based methods. To improve query accuracy while maintaining reasonable space consumption, reconstruction-based solutions are proposed. The main idea is that, instead of materializing the index for all possible ranges, a subset of ranges is selected to index. Specifically, a segment tree is employed for this selection purpose (see Fig. 3). Initially, the entire range $[1, N]$ is selected as the root and indexed. Subsequently, the range is divided into subranges (for simplicity, we use fanout 2 as an example in this paper). This division continues until the number of points within a given subrange falls below a predefined threshold. In total, the segment tree indexes $O(N \log N)$ ranges, which is a subset of the $O(N^2)$ possible ranges. Unlike compression-based solutions, when a query with a range $[l, r]$ is issued, the segment tree can reconstruct the index even if the query range is not explicitly recorded in the tree.

As shown in Fig. 3, when a query range $[l, r]$ is received, it is divided into multiple subranges that are already recorded in the tree. Thanks to the segment tree, this process of dividing the query range into recorded subranges can be performed efficiently. Subsequently, the graph indexes for these recorded subranges are combined to generate the index for the query range, thereby improving accuracy through reconstruction. However, this process incurs a cost: the indexes for $O(\log N)$ recorded subranges must be combined to answer the RFAKNN query, which can lead to slower query speeds.

*Drawbacks.* Although designed for RFAKNN queries, current state-of-the-art methods [9, 45, 52] face several challenges (see also Table 1). For compression-based methods, while they enable rapid query responses, their lossy nature inherently compromises query accuracy. For reconstruction-based methods, although they dynamically refine the range to mitigate accuracy loss, they require dividing the query into $O(\log N)$ subranges, which slows down the query process. These drawbacks motivate the study of our work.

## 3 Theoretical Findings

This section provides the theoretical findings that underpin our new methods for RFAKNN queries.

### 3.1 From PreFiltering to PostFiltering

As introduced in the previous section, state-of-the-art methods fall short in balancing query accuracy and query speed. Interestingly, we observe that both compression-based and reconstruction-based methods rely on the PreFiltering principle: They work only on graph indexes created on the in-range points of the query range, while completely ignoring the out-of-range points. In light of this, we resort to the PostFiltering principle to explore its potential in addressing the drawbacks of existing methods. However, this transition is non-trivial as PostFiltering typically requires searching across all points in the database (corresponding to the full range $[1, N]$) for any given query range. This results in query inefficiency, as it often includes many points outside the query range. For this purpose, we provide a theoretical analysis of how the PostFiltering principle can be *enhanced* to ensure query efficiency without compromising query accuracy.

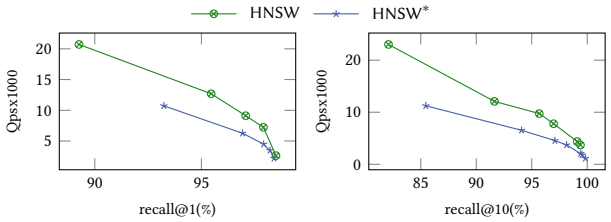



**Figure 4: The Illustration of the Potential of PostFiltering**

Example 2. *Figure 4 shows the effectiveness of utilizing* PostFiltering *when half of the data points are out-of-range. For this evaluation, we use* HNSW *to construct the graph index and assess its performance on the SIFT dataset. For processing RFAKNN queries, the* HNSW *method, represented by the green line, builds the index exclusively from in-range points. Conversely, the* HNSW* *method constructs an* HNSW *index for all points and integrates the* PostFiltering *method during search operations. The results reveal that* HNSW* *with* PostFiltering *achieves performance levels comparable to* HNSW*, particularly at high recall levels. Even at moderately lower recall levels, the performance difference remains minimal, within a twofold range.*

**Elastic Factor.** For PostFiltering, for any query range $[l, r]$, it operates on the graph containing all points (i.e., using only *one* range $[1, N]$) in $\mathcal{D}$. This is the root cause of PostFiltering being slow: it must process a large number of out-of-range points. To address this limitation, we propose relaxing this requirement by assuming that graph indexes are created for multiple ranges, denoted as $R$. We will analyze the performance of PostFiltering when multiple ranges are applied. The relationship between a range $[l, r]$ and the set of ranges $R$ is captured using the concept of Elastic Factor.

Definition 1 (Elastic Factor). *Given a set of ranges $R$, where each range $[l_i, r_i] \in R$ has $l_i, r_i \in [1, N]$, the elastic factor of a range $[l, r]$, with $l, r \in [1, N]$, is defined as:*

$$e(R, [l, r]) = \max_{[l,r] \subset [l_i, r_i]} \left( \frac{|[l, r]|}{|[l_i, r_i]|} \right).$$

*Here, $|[l, r]| = r - l + 1$ represents the length of a range $[l, r]$.*

The elastic factor $e(R, [l, r])$ can be understood as selecting the closest superset of a query range $[l, r]$ from $R$ to measure the ratio. This concept is similar to the blowup factor [9], but it focuses on how the range set covers the query range. Moreover, current studies on PostFiltering only consider when $R$ include one singe range $[1, N]$, which results in a very small elastic factor $e(R, [l, r])$.

## 3.2 Findings

Based on the elastic factor, we provide two key findings. **Our first theoretical finding** is that when $R$ includes a superset range $[a, b]$ of the query range $[l, r]$, the query accuracy is not compromised. This observation is intuitive: when $[a, b] \in R$ is a superset of $[l, r]$, all in-range points for the query range $[l, r]$ are also included in the range $[a, b]$. This implies that the graph index created for the query range $[l, r]$ is a subgraph of the superset range $[a, b]$. According to the properties of the monotonic search network, KNN searches on the subgraph can also be performed within the superset graph. Thus, we confirm this finding. This observation also explains why creating a graph index for range $[1, N]$ alone is sufficient for PostFiltering, as $[1, N]$ serves as a superset for all possible query ranges.

**Our second theoretical finding** is that when $e(R, [l, r])$ is sufficiently large (as $R$ can now include more ranges), the query speed of PostFiltering can be improved without sacrificing accuracy. Below is the formal proof of this claim.

**Complexity When $k$ is 1.** To investigate the query complexity of PostFiltering under elastic factor constraints, we begin by examining the query complexity of existing graph search methods (e.g., Algorithm 1) without range filtering, with a particular focus on the time complexity of $k$-nearest neighbor (KNN) search. In the existing literature, the special case of $k = 1$ has been extensively studied. Specifically, these methods construct graph indexes that satisfy the Monotonic Search Network (MSNET) property using edge-pruning strategies, ensuring that each point in the graph is associated with a monotonic search path.

Definition 2 (Monotonic Search Path [11]). *Let $\mathcal{D}$ be a database of $N$ points in $\mathbb{R}^d$, and let $G$ be a graph defined on $\mathcal{D}$. For any two points $p, q \in \mathcal{D}$, let $v_1, v_2, \ldots, v_k$ denote a path from $p$ to $q$ in $G$. This path is called a monotonic path if and only if $\forall i \in [1, k-1]$, $||v_i - q|| > ||v_{i+1} - q||$.*

Building on the concept of a monotonic search path, a Monotonic Search Network is defined as follows:

Definition 3 (Monotonic Search Network [11]). *Given a database $\mathcal{D}$ of $N$ points in $\mathbb{R}^d$ and a graph $G$ defined on $\mathcal{D}$, the graph $G$ is called a Monotonic Search Network if and only if there exists a monotonic search path between every pair of points $p, q \in \mathcal{D}$.*

From the above definitions, the query complexity of KNN search when $k$ is 1 is derived [11]:

$$O\left(\frac{N^{\frac{1}{d}} \log N^{\frac{1}{d}}}{\triangle r}\right), \tag{1}$$

Here, $\triangle r$ represents the minimal distance required to move closer to the query, while the degree of the graph index is bounded by a constant [11]. This query complexity can be interpreted as the expected length of the monotonic search path within the graph.

**Complexity for General $k$.** Equation 1 only accounts for the complexity where $k = 1$, leaving the query complexity for other cases unexplored. To bridge this gap, we offer the following conclusion:

Theorem 1. *Under the same assumptions as in [11], the expected search path length of MSNET with reverse edges for KNN search is:*

$$O\left(\frac{N^{\frac{1}{d}} \log N^{\frac{1}{d}}}{\Delta r} + k\right). \tag{2}$$

**Linking with Range Constraints.** Next, we focus on the query complexity of PostFiltering for range-filtering $k$-nearest neighbor (RFKNN) search, as this paper addresses scenarios where queries involve a range filter. We show that the RFKNN search problem can be identically solved by performing an $h$-nearest neighbor search, where $h$ is a value no smaller than $k$.

Lemma 1. *Given a database $\mathcal{D}$ of $N$ points in $\mathbb{R}^d$ with an additional numerical attribute, a query $q$ with a filter $[l, r]$, and the $h$-nearest neighbor set $S_h$ of $q$, let $S_k$ represent the range-filtering $k$-nearest neighbor set of $q$. If $|\{v_i \in S_h \mid i \in [l, r]\}| \geq k$, then $S_k \subseteq S_h$.*

Lemma 1 shows that the range-filtering $k$-nearest neighbor set, denoted as $S_k$, can be fully contained within the $h$-nearest neighbor set $S_h$ of $q$, provided that the number of in-range points in $S_h$ is no smaller than $k$. Building on this observation, an existing graph index can be utilized to perform an incremental $h$-nearest neighbor search. The process continues until $S_h$ contains at least $k$ in-range points, at which point $S_h$ is returned as the result for the RFKNN search. If this condition is not met, the search progresses incrementally to include the $(h+1)$-neighbor, which is how PostFiltering works.

**Complexity Under Range Constraints.** Since the query complexity for KNN search is already provided in Equation 2, the next challenge is estimating the value of $h$, which determines the query complexity for RFKNN search (by substituting $k$ with $h$ in the equation). In the worst case, $h$ could equal $N$, as it may be necessary to process all data points to report the result for range-filtering $k$-nearest neighbor search. However, by incorporating the elastic factor, we can prove that the expected value of $h$ is bounded by $\frac{k}{c}$, where $c$ is a value no smaller than the elastic factor of the query range $[l, r]$. This result ensures that the complexity of the RFKNN search remains comparable to the original KNN search, with only a proportional factor introduced by the range-filtering mechanism.

Theorem 2. *Given a database $\mathcal{D}$ of $N$ points, a range set $R$ (where each range is built using an MSNET), and a query $q$ with a filter range $[l, r]$. Assuming the conditions outlined in [11] and that the attribute values of points in $\mathcal{D}$ are independent, if the elastic factor of $[l, r]$ satisfies $e(R, [l, r]) \geq c$ for some constant $c$, the MSNET with reverse edges returns the range-filtering $k$ nearest neighbors with an expected search path length:*

$$O\left(\frac{N'^{\frac{1}{d}} \log N'^{\frac{1}{d}}}{\triangle r'} + k/c\right) \tag{3}$$

*where $N' = |[l, r]|/c$ and $\triangle r'$ is the minimal distance to get closer to the query.*

Theorem 2 shows that PostFiltering for RFKNN queries exhibits time complexity comparable to the optimal approach when the elastic factor is sufficiently large. Specifically, the time complexity described in Equation 3 can be reformulated as $O(N'^{\frac{2}{d}} \log N')$ with a probability of at least $1 - (1/e)^{\frac{d}{4}(1-\frac{3}{e^2})}$ with the conclusion in [35]. The size of the index range, $N'$, is at most $1/c$ of $N''$, which implies that the time complexity increases only by a constant factor. Consequently, it can be further reduced to $O(N''^{\frac{2}{d}} \log N'')$. The next challenge lies in designing a method to generate the range set $R$ such that the elastic factor exceeds a constant threshold $c$ (e.g., 0.5) for any query range.

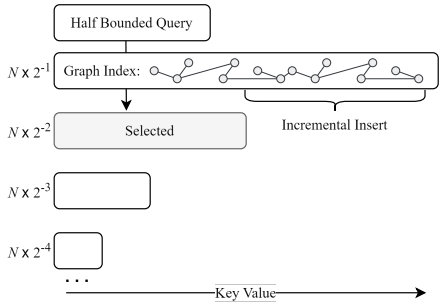



**Figure 5: The Example of $\mathsf{ESG}_{1D}$**

## 4 Our RFAKNN Query Processing Methods

In the previous section, we provided a theoretical analysis to ensure that, under an elastic factor no smaller than $c$, the search method based on the PostFiltering principle remains efficient. In this section, we propose novel reconstruction-based methods that fully leverage this conclusion to process RFAKNN queries efficiently. We first design a solution for the half-bounded RFAKNN search, and then we discuss the solution for the general RFAKNN search case. Note that our initial focus is on the case where $c = \frac{1}{2}$ and the more general case is discussed as an extension.

### 4.1 The Method for Half-Bounded Queries

To make it easy to understand, we first consider half-bounded RFAKNN queries. A half-bounded query means that the query range takes the form of either $[1, r]$ or $[r, N]$, where $r \in [1, N]$. This occurs when either the left bound is fixed at 1 or the right bound is fixed at $N$. For simplicity, we discuss the case of $[1, r]$, as the case of $[r, N]$ is similar. To answer the half-bounded RFAKNN query, an intuitive approach is to create $N$ graphs, one for each range $[1, i]$, where $i \in [1, N]$. However, due to the elastic factor, it is sufficient to create only $O(\log N)$ graphs. This results in the $\mathsf{ESG}_{1D}$ index, which we define as follows.

*Definition 4.1.* Given a database $\mathcal{D}$ of $N$ points, the $\mathsf{ESG}_{1D}$ index contains $\log N$ graphs. Each graph is constructed for the range $[1, N/2^i]$, where $i \in [0, \log N]$.

*Example 4.2.* Fig. 5 shows $\mathsf{ESG}_{1D}$, which consists of $O(\log N)$ graphs, each corresponding to range $[1, N/2^i]$, where $i \in [0, \log N]$.

**Query Processing.** To process an RFAKNN query with point $q$ and range $[1, r]$, we retrieve the $\mathsf{ESG}_{1D}$ index to locate the recorded graph under the range $[1, N/2^{\lceil \log r \rceil}]$. Then, we utilize PostFiltering to search this graph to find the answer. The rationale and justification for using this range, $[1, N/2^{\lceil \log r \rceil}]$, are as follows:

Lemma 4.3. $[1, r] \subseteq [1, N/2^{\lceil \log r \rceil}]$, *and* $\frac{|[1,r]|}{|[1,N/2^{\lceil \log r \rceil}]|} \geq 0.5$.

In Lemma 4.3, the condition that $[1, r] \subseteq [1, N/2^{\lceil \log r \rceil}]$ ensures that the range $[1, N/2^{\lceil \log r \rceil}]$ is a superset of $[1, r]$, which guarantees the query accuracy. The condition $\frac{|[1,r]|}{|[1,N/2^{\lceil \log r \rceil}]|} \geq 0.5$ ensures that the elastic factor of range $[1, r]$ is at least 0.5, which supports efficient query response. From Lemma 4.3, we conclude that $[1, N/2^{\lceil \log r \rceil}]$ serves as an appropriate superset of $[1, r]$, balancing accuracy and efficiency for RFAKNN queries.

**Algorithm 2:** Build-1D-Index($\mathcal{D}, N$)

**Input:** point $u \in \mathcal{D}$, cardinality $N$
**Output:** index $\mathsf{ESG}_{1D}$

1 $R \leftarrow [1, N/2^i]$ where $i \in [0, \log N]$; // init ranges
2 $\mathsf{ESG}_{1D} \leftarrow \emptyset, \mathcal{I} \leftarrow \emptyset$; // init index
3 $pre \leftarrow 1$;
4 **for** $cur \leftarrow 1$ ***to*** $N$ **do**
5  **if** $[1, cur] \in R$ **then**
6  insert $v_i$ into the graph $\mathcal{I}$, $i \in [pre, cur]$;
7  $\mathsf{ESG}_{1D} \leftarrow \mathsf{ESG}_{1D} \cup (\mathcal{I}, [1, cur])$;
8  $pre \leftarrow cur$;
9 **return** $\mathsf{ESG}_{1D}$;

**Index Construction.** Recall that it is feasible to construct the graph index separately for all ranges $R = \{[1, N/2^i] \mid i \in [0, \log N]\}$. However, we observe that when constructing the graph index for the range $[1, N]$, all other subranges in $R$ can be derived as byproducts. Specifically, by sorting the points in non-increasing order based on their attribute values, the index for any range $[1, cur]$ (where $cur \leq N$) is incrementally created until we reach $cur = N$ and the index for $[1, N]$ is created.

*Algorithm.* Algorithm 2 outlines the process for creating $\mathsf{ESG}_{1D}$. Initially, we determine the required ranges $R$ for the $\mathsf{ESG}_{1D}$ (Line 1). Next, we initialize both the index $\mathsf{ESG}_{1D}$ and the temporary index $\mathcal{I}$ as empty sets (Line 2). It then iterates over the data points in increasing numerical order (Line 4). During each iteration, we increment $cur$ by 1 and check if the range $[1, cur]$ is included in $R$ (Line 5). If yes, we update the temporary index $\mathcal{I}$ by inserting $v_i$, where $v_i$ is the data point newly included since the previous range $[1, pre]$. This step creates the graph index for the range $[1, cur]$. Then, we insert $\mathcal{I}$ along with its corresponding range $[1, cur]$ into $\mathsf{ESG}_{1D}$ (Line 7). Finally, we update $pre$ to $cur$ for the next iteration (Line 8). Once all points in $\mathcal{D}$ have been processed, it returns the constructed index $\mathsf{ESG}_{1D}$ (Line 9).

**Complexity Analysis.** We first study the index space of $\mathsf{ESG}_{1D}$. As the graph degree is bounded by a constant $M$, the graph nodes in $\mathsf{ESG}_{1D}$ can be computed by the sum of range length in $R$. Then the summation of $[1, N/2^i]$ can be bounded by $2N$ and we get that the index size of $\mathsf{ESG}_{1D}$ is $O(NM)$. Next, we analyze the indexing time, and from Line 4 of Algorithm 2, we know the creation of $\mathsf{ESG}_{1D}$ requires $O(N)$ insertion of HNSW index. The memory snapshot and disk storage only take a tiny portion of the overall time cost in practice. Finally, since it only requires scanning $O(\log N)$ graph indexes for a query, and the elastic factor is bounded by 0.5, then the query complexity is still bounded with the conclusion in Theorem 2.

**Extensions.** To achieve a more flexible elastic factor, $\frac{1}{B}$, greater or less than $0.5 = \frac{1}{2}$ for tradeoff efficiency and space, we construct graphs for the ranges $[1, N/B^i]$, where $i \in [0, \log_B N]$. Recall that a larger elastic factor, $\frac{1}{B}$, enables more efficient query processing but with more space cost. On the contrary, a smaller factor saves space but reduces efficiency. Therefore, the elastic factor acts as a tunable parameter to balance space cost and query time.

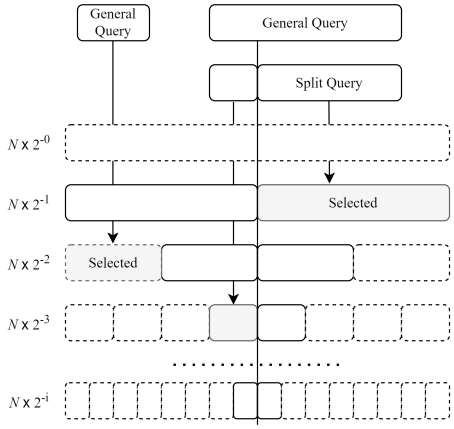



**Figure 6: The Example of $\mathrm{ESG}_{2D}$**

**Algorithm 3:** Build-2D-Index($\mathcal{D}$, $[l, r]$)

```
1 if r − l < ℓ then
2     return ∅ ;                    // no index for small range
3 mid = ⌊(l + r)/2⌋;
4 I_l=Build-2D-Index(D, [l, mid]);
5 I_r=Build-2D-Index(D, [mid+1, r]);
6 if I_l = ∅ then
7     create graph I_l on v_i, i ∈ [l, mid];
8 create graph I by inserting v_i into I_l, i ∈ [mid + 1, r];
9 ESG_2D ← ESG_2D ∪ (I, [l, r]);
```

## 4.2 The Method for General Queries

We now introduce the process for handling a general RFAKNN query. To facilitate this, we define the index structure, $\mathrm{ESG}_{2D}$. Similar to $\mathrm{ESG}_{1D}$, the $\mathrm{ESG}_{2D}$ index is also a hierarchical structure. Note that the $\mathrm{ESG}_{2D}$ index resembles those defined in current reconstruction-based methods for RFAKNN queries, where the hierarchical structure is encoded in a segment tree. However, the main technical contribution lies in how to accelerate index construction and leverage elastic factors to optimize the query process.

*Definition 4.4.* Given a database $\mathcal{D}$ of $N$ points, the $\mathrm{ESG}_{2D}$ index contains $\log N$ layers and is organized as a segment tree. The root of the tree corresponds to the range $[1, N]$, and its children are recursively defined as divisions of the range into subranges.

Initially, a graph is constructed for the range $[1, N]$. This range is then recursively divided into two subranges, $[1, N/2]$ and $[N/2+1, N]$, etc. The resulting structure forms a segment tree, where each node corresponds to a subrange of $[1, N]$, and a graph is constructed for the associated subrange. An example is given in Fig. 6.

**Indexing Algorithm.** Algorithm 3 shows how to create $\mathrm{ESG}_{2D}$. While Algorithm 3 generates the index similarly to existing reconstruction-based methods, it is more efficient as it aims to reduce redundancy and improve performance. Specifically, the algorithm takes the entire range $[1, N]$ as input and calls the recursive function (Algorithm 3) to build a segment tree. The recursion halts when the range becomes sufficiently small (Line 1). Otherwise, the current range $[l, r]$ is divided into two subranges, $[l, \mathrm{mid}]$ and $[\mathrm{mid}+1, r]$, and the recurision continues (Line 4-5). Then, if the graph index for the left subrange is not present, the algorithm incrementally inserts points $v_i$ from $i \in [l, \mathrm{mid}]$ to construct the graph index for that range (Line 6-7). The most notable aspect of the approach is that instead of directly creating the graph index for the range $[l, r]$, it leverages the graph already constructed for the left subrange (created in Line 4 or Line 7) and incrementally adds points from the right subrange $[\mathrm{mid}+1, r]$. This strategy avoids building the graph index from scratch, significantly enhancing efficiency.

**Algorithm 4:** Query-2D-Index($\mathrm{ESG}_{2D}$, $q$, $[l, r]$, $[l_q, r_q]$)

```
1 if r − l < ℓ then
2     return linear scan of v_i, i ∈ [l, r];
3 if [l_q, r_q] ⊆ [l, r] and |[l_q,r_q]|/|[l,r]| ≥ c then
4     I ← ESG_2D(l, r) ;        // select index based on range
5     return graph search of q, [l, r] using I
6 mid = ⌊(l + r)/2⌋;
  /* search left sub-tree                                  */
7 if l_q ≤ mid then
8     S_l=Query-2D-Index(ESG_2D, [l, mid], [l_q, min(mid, r_q)]);
  /* search right sub-tree                                 */
9 if r_q > mid then
10    S_r=Query-2D-
        Index(ESG_2D, [mid+1, r], [max(mid+1, l_q), r_q])
11 return merge sort result of S_l and S_r;
```

**Query Algorithm.** Algorithm 4 shows how to use $\mathrm{ESG}_{2D}$ to process a query with point $q$ and range $[l_q, r_q]$. It follows a recursive process, with the termination condition being when the selected subrange $[l, r]$ becomes smaller than a predefined threshold. At this point, a linear scan is performed to report the answer (Line 1-2). If the query range is $c$ times larger than the selected subrange $[l, r]$ (where $c$ is the elastic factor), the graph stored in $\mathrm{ESG}_{2D}$ is utilized along with PostFiltering to find the answer (Line 3-5). Otherwise, the subrange (initially set to $[1, N]$) is divided into two parts (Line 6), and the algorithm is applied recursively to each part: the left part (Line 7-8) and the right part (Line 9-10). The results from the left and right parts are then merged to produce the final answer (Line 11). Algorithm 4 is similar to existing reconstruction-based methods. Yet, a key distinction is that when a subrange encountered during traversal does not exactly match the current query range, it can still be used if Line 3 satisfies, thanks to the use of the elastic factor.

**Query Complexity.** One might wonder whether the recursion continues until reaching the leaf node of the segment tree. Yet, the number of graph indexes that $\mathrm{ESG}_{2D}$ needs to select is limited to 2.

Lemma 2. *Given an RFAKNN query, Algorithm 4 selects at most two graph indexes in the worst case, assuming a fanout of 2.*

Lemma 2 highlights the advantages of our new method for the RFAKNN query. Specifically, if the query range is fully contained within a superset range recorded in the segment tree, and the elastic factor constraint is satisfied, the graph index corresponding to the current superset range can be directly utilized for the PostFiltering search. This requires only a single graph index. In cases where the

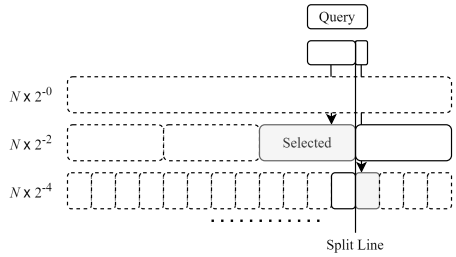



**Figure 7: The Examples of a Larger Fanout**

query range is not fully contained, it is split just once. As a result, the query complexity remains effectively bounded by Theorem 2.

*Example 4.5.* In Fig. 6, when a query with the range $[l, r]$ is received, and it is fully contained within an indexed range $[a, b]$, we directly select $[a, b]$ if the size of $[l, r]$ is no smaller than that of $[a, b]$, adjusted by the elastic factor of 0.5. In contrast, the state-of-the-art method requires finding subranges that exactly cover $[l, r]$. In cases where the above condition is not satisfied, we split the range $[l, r]$ into two parts, ensuring that each part falls within a recorded range $[a, b]$ that adheres to the elasticity factor of 0.5.

**Index Cost Analysis.** We begin by analyzing the space cost of $\mathsf{ESG}_{2D}$. Given that the graph degree is bounded by a constant $M$, the graph nodes in $\mathsf{ESG}_{2D}$ can be computed as the sum of the range lengths in $R$. Since the ranges in $R$ are organized using a segment tree with $O(\log N)$ layers, and each layer consists of $N$ points, the total range length in $R$ is bounded by $N \log N$. Thus, the index size of $\mathsf{ESG}_{2D}$ is $O(NM \log N)$. Next, we examine the indexing time. Based on the space cost analysis, the total number of nodes in $\mathsf{ESG}_{2D}$ is bounded by $O(N \log N)$. Algorithm 3 requires $O(N \log N)$ insertions into the HNSW index. Note that, in practice, the number of insertions in Algorithm 3 is approximately half the total number of nodes in $\mathsf{ESG}_{2D}$. This reduction occurs because the incremental construction process utilizes the left subtree, allowing each range with a left subtree to save half of the insertion time.

**Extensions.** In prior discussions, we fixed the fanout to 2 and set the elastic factor to 0.5. Then, we explore the extensions of a segment tree with a larger fanout and analyze their impact on both space and time complexity. Increasing the fanout, for instance, to 4, results in reductions in both indexing time and index space. This improvement arises as the original $\log_2(N)$ layers of the tree are reduced to $\log_4(N)$ layers, halving the costs associated with index construction in terms of both time and space. These improvements are illustrated in Fig. 7. To ensure consistency with our previous findings (see Lemma 2), we observed that setting the elastic factor constraint to the reciprocal of the fanout ensures that the time complexity remains bounded. This strategy guarantees that, in the worst case, only two indexes are required.

Lemma 3. *Let $f$ denote the fanout of the segment tree and $1/f$ as the elastic factor constraint, then Algorithm 4 also selects two graph indexes in the worst case.*

Example 3. *Figure 7 shows $\mathsf{ESG}_{2D}$ with a fanout of 4 and an elastic factor constraint of 1/4. The uniform range split ensures that the query range either meets the 1/4 elastic factor constraint or spans only two sub-tree index ranges. If the query range requires three or more subranges for coverage, the middle index range is fully enclosed within the query range, thereby satisfying the elastic factor constraint. As a result, the original general query can still be transformed into two half-bounded queries.*

Regarding the impact on query complexity, let $f$ be the fanout of the segment tree, which is a constant. The elastic factor is constrained to be $\geq 1/f$, where the probability of success (i.e., finding an in-range point) is $1/f$. From Theorem 2, the expected number of search steps is given by $E[W_k] = k \times \frac{N+1}{Nf+1}$, where $Nf$ denotes the number of *in-range* points. For cases where $f$ is $1/4$ or $1/8$, the additional search steps increase by a factor of 4 and 8, respectively. However, under the assumption that $f$ remains a fixed constant and $f < N$, the extra steps are still bounded by a constant factor of $k$. Consequently, the overall time complexity remains unaffected.

# 5 Experiments

## 5.1 Experiment Settings

**Datasets.** We use publicly available datasets that are widely used as benchmarks for RFAKNN search: SIFT, GLOVE, WIT, and DEEP100M. Among these, DEEP100M is a large-scale dataset comprising 100 million instances sampled from DEEP1B[2], which we use to evaluate scalability. The WIT dataset[3], which includes 2048-dimensional ResNet-50 embeddings of images from Wikipedia, was further processed by using the size of each image as the attribute value. All datasets used in our experiments are stored in the float32 format. For datasets that do not provide attribute values, we synthesize these attributes following the methods described in [45, 52]. Details of all datasets, including the number of data points ($\mathcal{N}$), dimensionality, and query size, are summarized in Table 3. To form query ranges, we randomly selected two values from the range $[1, N]$ as the left and right bounds of the query range for general RFAKNN queries. For half-bounded queries, only one value was selected from $[1, N]$. We refer to these randomly generated queries collectively as "range = mix". Moreover, to evaluate the impact of query range lengths on performance, we also vary range lengths as $2^{-1} \times N$, $2^{-3} \times N$, and $2^{-8} \times N$.

**Metrics.** To evaluate the accuracy of our method, we use recall as the metric due to its extensive use in benchmarks [2, 42]. For the efficiency evaluation, we use the queries-per-second (QPS), which can be regarded as the number of queries that the algorithm can process in one second. All metrics used in the experiment are reported on averages over the query set.

**Algorithms.** The algorithms compared in our study are as follows:
- $\mathsf{ESG}_{1D}$: Our proposed method for Half-Bound RFAKNN queries.
- $\mathsf{ESG}_{2D}$: Our proposed method for General RFAKNN queries.
- $\mathsf{SeRF}_{1D}$: Compression-based method for Half-Bound queries [52].
- $\mathsf{SeRF}_{2D}$: Compression-based method for General queries in [52].
- SegmentTree: Reconstruction-based method in [9].
- Super: The SuperPostFiltering approach [9].
- IRANGE: Another reconstruction-based method in [45].

[2]https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search
[3]https://github.com/google-research-datasets/wit

**Table 3: The Statistics of Datasets**

| Dataset | Dimension | Size | Query Size | Type |
|---|---|---|---|---|
| SIFT | 128 | 1,000,000 | 1000 | Image + Attribute |
| DEEP | 96 | 1,000,000 | 1000 | Image + Attribute |
| GLOVE | 100 | 1,000,000 | 1000 | Text + Attribute |
| WIT | 2048 | 1,000,000 | 1000 | Image + Attribute |
| DEEP100M | 96 | 100,000,000 | 1000 | Image + Attribute |

**Table 4: The Comparison of Index Time (s)**

| Dataset | $\text{SeRF}_{1D}$ | $\text{ESG}_{1D}$ | $\text{SeRF}_{2D}$ | $\text{ESG}_{2D}$ | Super | IRANGE |
|---|---|---|---|---|---|---|
| SIFT | 11 | 14 | 18 | 54 | 142 | 475 |
| DEEP | 9 | 12 | 17 | 60 | 138 | 463 |
| GLOVE | 7 | 12 | 14 | 80 | 332 | 431 |
| WIT | 84 | 129 | 163 | 297 | 1129 | 2,250 |
| DEEP100M | 1,518 | 2,007 | 3,910 | 10,554 | - | >24h |

**Table 5: The Comparison of Index Size (MB)**

| Dataset | Raw Data | $\text{SeRF}_{2D}$ | $\text{ESG}_{2D}$ | Super | IRANGE |
|---|---|---|---|---|---|
| SIFT | 489 | 176 | 1,416 | 2,549 | 878 |
| DEEP | 367 | 216 | 1,416 | 2,549 | 957 |
| GLOVE | 386 | 129 | 1,416 | 2,549 | 664 |
| WIT | 7,812 | 154 | 1,416 | 2,549 | 761 |
| DEEP100M | 36,621 | 22,012 | 226,630 | - | - |

**Implementation Details.** All code was written in C++ and compiled using GCC version 9.4.0 with -Ofast optimization. The experiments were conducted on a workstation equipped with Intel(R) Xeon(R) Platinum 8352V CPUs @ 2.10GHz, 1TB of memory, and running Ubuntu Linux. We utilized multi-threading for index construction and a single thread for search evaluation. Since both IRANGE and SeRF are based on HNSW, we used HNSW as the graph index for $\text{ESG}_{1D}$, $\text{ESG}_{2D}$, SegmentTree, and Super to isolate and focus on the algorithmic performance. The default fanout for $\text{ESG}_{2D}$ was set to 2, and we also evaluated the impact of larger fanout values in the sequel. The parameters of all existing algorithms were configured according to the default settings.

## 5.2 Experiment Results

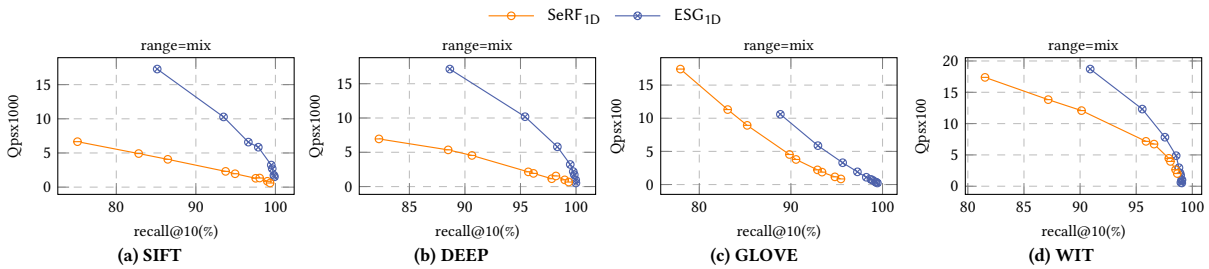
**Exp-1: Half-Bounded Query Performance.** We begin by evaluating the performance of different algorithms on processing half-bounded RFAKNN queries. For this analysis, $\text{SeRF}_{1D}$ and $\text{ESG}_{1D}$, both of which require only $O(N)$ space complexity, are used as comparison methods. Our method, $\text{ESG}_{1D}$, is compared against $\text{SeRF}_{1D}$ across randomly generated query ranges (denoted as range = mix). The results, shown in Fig. 8, where the top-right region indicates better performance, lead to the following observations:
• Our $\text{ESG}_{1D}$ algorithm consistently outperforms $\text{SeRF}_{1D}$ across nearly all datasets, achieving average search efficiency improvements of 1.2x to 2x. This superior performance can be attributed to two main factors: 1) The integration of well-optimized search libraries within the $\text{ESG}_{1D}$ framework enhances overall efficiency. 2) The hierarchical structure and memory layout of HNSW are preserved in $\text{ESG}_{1D}$, which not only minimizes engineering development costs but also significantly boosts search performance.

**Exp-2: General Query Performance.** We further investigate the performance of various methods for general RFAKNN queries. To evaluate their performance, we designed range queries of varying lengths, spanning from $2^{-1}$ to $2^{-8}$ of $N$. The primary methods compared include $\text{ESG}_{2D}$, SegmentTree, $\text{SeRF}_{2D}$, Super, and IRANGE. Notably, the SegmentTree algorithm utilizes the same index as $\text{ESG}_{2D}$ but employs a different query algorithm. Based on the results in Fig 9, we observe the following findings:
• $\text{ESG}_{2D}$ achieves comparable search efficiency to Super at high recall levels across all datasets. Specifically, $\text{ESG}_{2D}$ outperforms Super on the GLOVE and WIT datasets and demonstrates strong competitiveness with Super on the SIFT and DEEP datasets.
• Our $\text{ESG}_{2D}$ outperforms IRANGE on the SIFT, DEEP, and GLOVE datasets, achieving up to a 2x improvement in query efficiency in scenarios with smaller range filters and high recall levels. However, we observe some performance gaps relative to IRANGE on the WIT dataset and in scenarios with ranges with large lengths. This discrepancy stems from the need for our $\text{ESG}_{2D}$ to compute distances to certain points outside the range. A potential solution to address this limitation is to incorporate distance computation acceleration techniques [7, 13, 14, 30, 47], which will be our future research.

**Exp-3: Index Time and Space.** We compare the index time and space for various methods in Table 4. First, we observe that the indexing time of our ESG is comparable to that of SeRF, whereas IRANGE requires 5x to 10x more time compared to ESG, and Super demands 2.7x to 4x the indexing time. Two primary factors contribute to this efficiency: 1) Our method leverages a well-optimized algorithm library, which ensures better concurrency and computational efficiency. 2) It capitalizes on redundant information during the construction process by employing incremental construction based on the left subtree index, thereby further reducing the overall construction time. Next, we examine the index space. As shown in Table 5, our method achieves an index size comparable to $\text{SeRF}_{2D}$ and IRANGE, while demonstrating a smaller space cost compared to Super. Notably, Super requires nearly double the space of $\text{ESG}_{2D}$, particularly in scenarios involving lower-dimensional data.

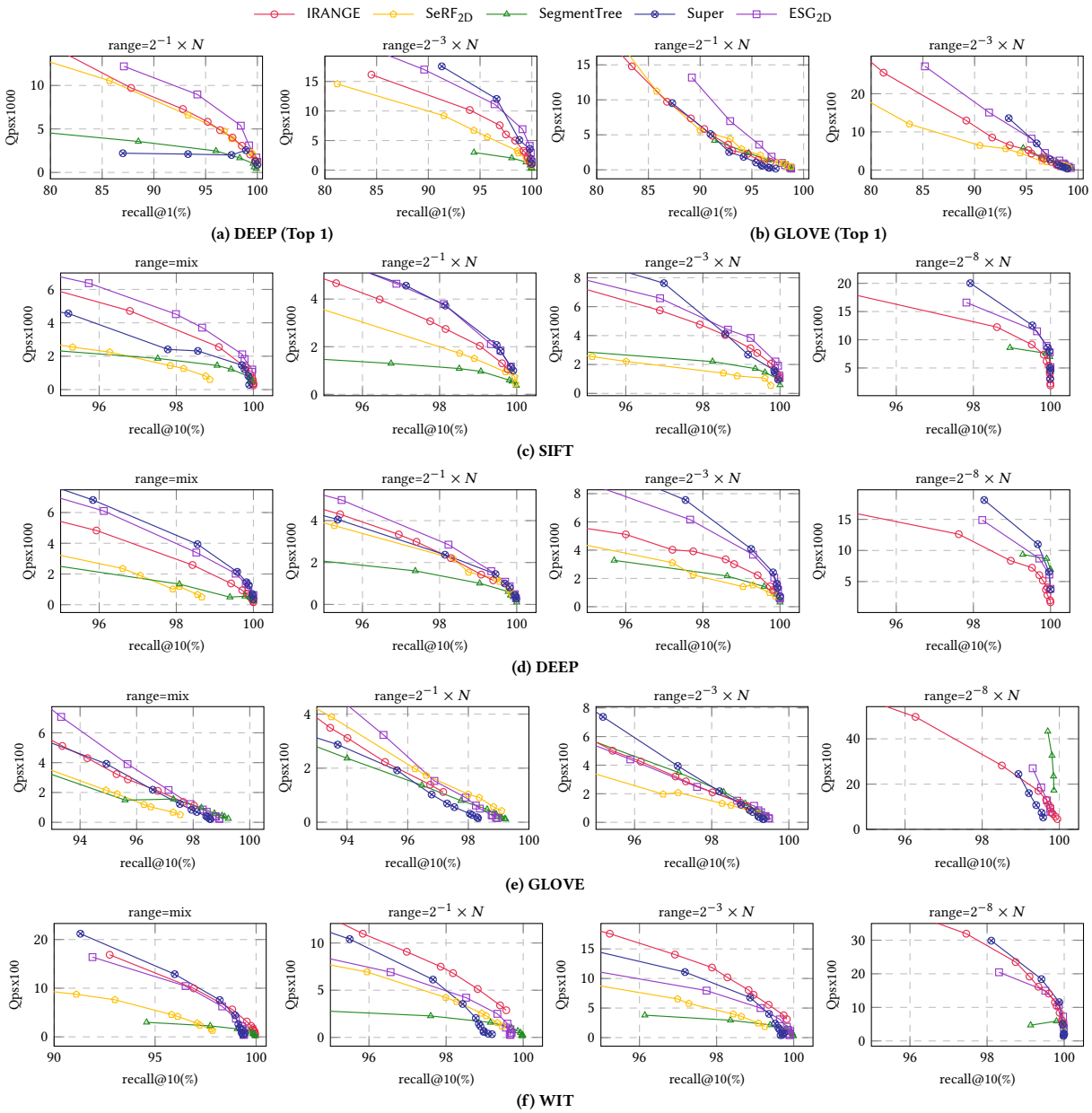
**Exp-4: Test of Scalability.** We further evaluate the scalability of different methods. To this end, we conduct experiments on the largest dataset, DEEP100M, varying the recall rate to compare different methods. As shown in Fig. 10, SERFO shows a 6x lower performance compared to our proposed $\text{ESG}_{1D}$ at 98% recall. Also, $\text{ESG}_{2D}$ maintains stable performance on large-scale datasets, whereas $\text{SeRF}_{2D}$ fails to achieve the target recall of 98%. Our method also offers significant advantages in space efficiency and indexing time on large-scale datasets by controlling the fanout size. For instance, with a fanout of 16 (the efficiency result in appendix), $\text{ESG}_{2D}$-16 requires only 70GB of storage—just one-third of the space required for a fanout of 2—and 5830 seconds of construction time.

**Exp-5: Test of Top-1 Search.** We compare different methods of processing range-filtering approximate nearest neighbor search (where $k = 1$). As shown in Fig. 9(a) and (b), our proposed method, $\text{ESG}_{2D}$, consistently achieves a 1.3x to 2x improvement compared to the IRANGE and Super. Our theoretical analysis reveals that the number of steps required to identify the global nearest neighbor scales sublinearly with the dataset size, while the additional search steps increase linearly with $k$. This provides a dramatic advantage for our method in RFANN search.




(a) SIFT (b) DEEP (c) GLOVE (d) WIT

**Figure 8: The Comparison of Performance on Processing Half-Bound RFAKNN Queries**




(a) DEEP (Top 1) (b) GLOVE (Top 1)

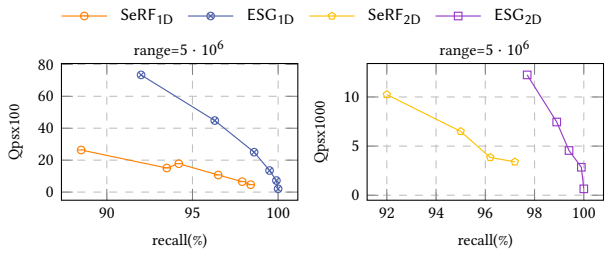
(c) SIFT

(d) DEEP

(e) GLOVE

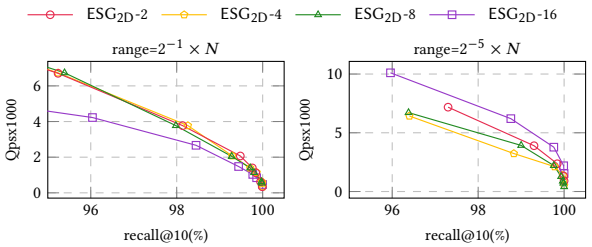
(f) WIT

**Figure 9: The Comparison of Performance on Processing General RFAKNN Queries**




(a) DEEP100M (Half) (b) DEEP100M (General)

**Figure 10: The Test of Scalability**




(a) DEEP (Varify Fanout)

**Figure 11: The Test of a Large Fanout**

**Exp-6: Test of Large Fanout.** We set the fanout to 2 for the segment tree in our method, $\text{ESG}_{2D}$. Next, we test $\text{ESG}_{2D}$ when it is configured with a larger fanout. As shown in Fig. 10, a larger fanout provides the benefit of more cheap index space. For instance, on the DEEP dataset, $\text{ESG}_{2D}$-4, $\text{ESG}_{2D}$-8, and $\text{ESG}_{2D}$-16—corresponding to fanouts of 4, 8, and 16—require only 709MB, 567MB, and 426MB of space, respectively. We further analyze search efficiency in Fig. 11. The results indicate that $\text{ESG}_{2D}$-2 achieves relatively good overall query efficiency. However, it does not exhibit a big performance advantage compared to configurations with larger fanouts, which is consistent with our theoretical analysis. Additionally, $\text{ESG}_{2D}$-16 even shows some advantages for smaller range filters. This is primarily due to the more relaxed elastic factor constraint (>1/16), which increases the likelihood of using a single index to efficiently answer general RFANN search queries.

## 6 Related Work

**Attribute-filtering AKNN Search.** Attribute-filtering approximate $k$ Nearest Neighbor (AFAKNN) search represents a more general case of RFAKNN search. While RFAKNN focuses solely on a single numeric attribute, AFAKNN incorporates additional attribute values, such as discrete labels, to broaden its applicability. Existing studies [20, 40, 44, 49] address AFAKNN search from both database systems and algorithmic perspectives [9]. 1) From the database systems perspective, query cost prediction is a core strategy for allocating appropriate search methods to AFAKNN queries. Systems such as ADB [44], Vbase [49], and Milvus [39] exemplify this approach. 2) On the algorithmic side, several methods aim to enhance query efficiency through attribute-specific optimizations. Filtered-DiskANN [17] improves efficiency by predicting attribute values, NHQ [40] refines the graph-index structure based on attribute values, and HQI [32] processes queries offline by leveraging the workload. The recent work UNG [6] proposes a unified approach to AFAKNN search by constructing a label navigating graph for attribute values. Despite these advancements, there remains a significant performance gap between AFAKNN algorithms and specialized RFAKNN search algorithms, such as SeRF, IRANGE, and SuperPostFiltering [9, 45, 52]. As such, our baseline evaluation focuses exclusively on highly competitive algorithms like SeRF, IRANGE, and SuperPostFiltering to ensure robust and meaningful comparisons.

**Optimizing Segment Tree.** Current reconstruction-based methods for RFAKNN search queries primarily rely on segment trees, which require $O(\log N)$ subranges during query processing. To reduce the number of required subranges, the SuperPostFiltering algorithm proposed in [9] redundantly stores certain ranges within the index. However, this approach comes with a significant trade-off: it incurs nearly double the space overhead compared to partitioning with segment trees. As an empirical analysis, our proposed $\text{ESG}_{2D}$ method is competitive with SuperPostFiltering and demonstrates superior performance in top-1 search (when $k = 1$ for RFAKNNAKNN search). In addition, our work provides a theoretical analysis of the PostFiltering algorithm, establishing that at most 2 subranges are required during query processing. This robust theoretical foundation highlights the novelty of our approach in addressing RFAKNN search queries effectively.

**Similarity Search in General Spaces.** In addition to Euclidean distance, other distance metrics such as inner product and angular distance are widely used in high-dimensional AKNN search. When the inner product is selected as the distance metric, it forms the problem of Maximum Inner Product Search (MIPS), which can be efficiently addressed using LSH-based methods [37, 46, 50] with theoretical guarantees. Notably, graph-based indexes are also extensively applied to MIPS [28, 33], often demonstrating superior search performance compared to LSH-based approaches. In this paper, we propose methods based on the PostFiltering principle within Euclidean space, accompanied by rigorous theoretical analysis. The core idea is to filter out range-restricted k-nearest neighbors (KNN) from the global KNN search. We also aim to extend this approach to handle scenarios where the inner product is used as the distance metric, enabling solutions for more general distance or similarity search problems.

## 7 Conclusions

We focus on the problem of RFAKNN queries by introducing a paradigm shift from the commonly used PreFiltering principle to the PostFiltering principle. This transition enables a theoretical analysis within a more generalized query complexity framework for PostFiltering, particularly when incorporating the elastic factor. Building upon this theoretical foundation, we propose novel reconstruction-based methods that guarantee at most two subranges are required for a query, compared to the $O(\log N)$ subranges employed by existing approaches. Extensive experimental results demonstrate the superiority of our methods in query efficiency while preserving query accuracy. In future work, we aim to extend this concept to other index structures, focusing on enhancing construction efficiency and optimizing space utilization.

## References


[1] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2015. Cache locality is not enough: High-Performance Nearest Neighbor Search with Product Quantization Fast Scan. *Proceedings of the VLDB Endowment* 9, 4 (2015).

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

[3] Artem Babenko and Victor S. Lempitsky. 2015. The Inverted Multi-Index. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 6 (2015), 1247–1260.

[4] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*. 97–104.

[5] Davis W Blalock and John V Guttag. 2017. Bolt: Accelerated data mining with fast vector compression. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 727–735.

[6] 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.

[7] Patrick Chen, Wei-Cheng Chang, Jyun-Yu Jiang, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. 2023. FINGER: Fast Inference for Graph-based Approximate Nearest Neighbor Search. In *Proceedings of the ACM Web Conference 2023*. 3225–3235.

[8] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[9] Joshua Engels, Benjamin Landrum, Shangdi Yu, Laxman Dhulipala, and Julian Shun. 2024. Approximate Nearest Neighbor Search with Window Filters. *ICML 2024* (2024).

[10] Cong Fu, Changxu Wang, and Deng Cai. 2022. High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 8 (2022), 4139–4150.

[11] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474.

[12] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 541–552.

[13] Jianyang Gao and Cheng Long. 2023. High-Dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations. *Proc. ACM Manag. Data* 1, 2 (2023), 137:1–137:27. https://doi.org/10.1145/3589282

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

[15] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 4 (2014), 744–755.

[16] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[17] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, et al. 2023. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*. 3406–3416.

[18] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. 2016. Quantization based fast inner product search. In *Artificial intelligence and statistics*. PMLR, 482–490.

[19] Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J Smola. 2022. BLISS: A Billion scale Index using Iterative Re-partitioning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 486–495.

[20] Gaurav Gupta, Jonah Yi, Benjamin Coleman, Chen Luo, Vihan Lakshman, and Anshumali Shrivastava. 2023. CAPS: A Practical Partition Index for Filtered Similarity Search. *arXiv preprint arXiv:2308.15014* (2023).

[21] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.

[22] 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.

[23] 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.

[24] 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).

[25] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128.

[26] Yannis Kalantidis and Yannis Avrithis. 2014. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2321–2328.

[27] 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.

[28] Jie Liu, Xiao Yan, Xinyan Dai, Zhirong Li, James Cheng, and Ming-Chang Yang. 2020. Understanding and improving proximity graph based maximum inner product search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 139–146.

[29] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: Hierarchical Graph Structure Based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 15, 2 (2021), 246–258.

[30] Kejing Lu, Chuan Xiao, and Yoshiharu Ishikawa. [n.d.]. Probabilistic Routing for Graph-Based Approximate Nearest Neighbor Search. In *ICML 2024*.

[31] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.

[32] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. 2023. High-throughput vector similarity search in knowledge graphs. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.

[33] Stanislav Morozov and Artem Babenko. 2018. Non-metric similarity graphs for maximum inner product search. *Advances in Neural Information Processing Systems* 31 (2018).

[34] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11 (2002), 28–46.

[35] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. *Proc. ACM Manag. Data* 1, 1 (2023), 54:1–54:27. https://doi.org/10.1145/3588908

[36] Maxim Raginsky and Svetlana Lazebnik. 2009. Locality-sensitive binary codes from shift-invariant kernels. *Advances in neural information processing systems* 22 (2009).

[37] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Advances in neural information processing systems* 27 (2014).

[38] 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. *Proc. VLDB Endow.* 8, 1 (2014), 1–12.

[39] 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.

[40] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2022. Navigable proximity graph-driven native hybrid queries with structured and unstructured constraints. *arXiv preprint arXiv:2203.13601* (2022).

[41] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *Proc. ACM Manag. Data* 2, 1 (2024), V2mod014:1–V2mod014:27. https://doi.org/10.1145/3639269

[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] Runhui Wang and Dong Deng. 2020. DeltaPQ: lossless product quantization code compression for high dimensional similarity search. *Proceedings of the VLDB Endowment* 13, 13 (2020), 3603–3616.

[44] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: a hybrid analytical engine towards query fusion for structured and unstructured data. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3152–3165.

[45] Yuexuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S Jensen. 2024. iRangeGraph: Improvising Range-dedicated Graphs for Range-filtering Nearest Neighbor Search. *arXiv preprint arXiv:2409.02571* (2024).

[46] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-ranging lsh for maximum inner product search. *Advances in Neural Information Processing Systems* 31 (2018).

[47] 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

[48] Qiang Yue, Xiaoliang Xu, Yuxiang Wang, Yikun Tao, and Xuliyuan Luo. 2023. Routing-Guided Learned Product Quantization for Graph-Based Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2311.18724* (2023).

[49] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, et al. 2023. {VBASE}: Unifying

Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 377–395.

[50] Xi Zhao, Bolong Zheng, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S Jensen. 2023. FARGO: Fast maximum inner product search via global multi-probing. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1100–1112.

[51] Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S. Jensen. 2020. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *Proc. VLDB Endow.* 13, 5 (2020), 643–655.

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

# A Appendix

## A.1 Proofs

**Proof of Lemma 1.** From the definition 2.2 of range-filtering $k$ nearest neighbor search, the range-filtering $k$ nearest neighbor set $S_k$ of a query $q$ with filter $[l, r]$ consists of the $k$ nearest neighbors within $[l, r]$, these points are, by definition, closer (in terms of the distance metric) than any other points within $[l, r]$ not in $S_k$. Given that $S_r$ is the set of the top-$r$ nearest neighbors overall, and it contains at least $k$ points within $[l, r]$, these $k$ points must be among the $r$ nearest overall points. Next, we prove this problem by contradiction. If there were a point in $S_k$ that was not in $S_r$, it would imply that there exists a closer point (since $S_r$ already includes the top $r$ nearest neighbors). However, this contradicts the assumption that $S_k$ contains the $k$ nearest neighbors within the range. Therefore, all points in $S_k$ must be included in $S_r$ to satisfy both the range condition and the nearest neighbor condition.

**Proof of Theorem 1.** The author of [11] discusses the search time complexity of nearest neighbor search. However, there is no direct statement of the top $k$ nearest neighbor search time complexity. Recall that with the same assumption in [11], the Algorithm 1 finds the top 1 nearest neighbor with the expected length

$$O(\frac{N^{\frac{1}{d}} \log N^{\frac{1}{d}}}{\triangle r}).$$

Then we get the top $k$ nearest neighbor search time complexity based on the reduction method and the property of MSNET **??**. Consider we find the top 1, ..., $k-1$ nearest neighbor $v_1, .., v_{k-1}$ of $q$ and search for the top $k$-th nearest neighbor $v_k$. From the property of MSNET, any two points have a monotonic search path. The $v_k$ and $q$ are connected by a monotonic search path with the same expected length. Then, the expected search path length of the top $k$ nearest neighbor search will be

$$O(k \times \frac{N^{\frac{1}{d}} \log N^{\frac{1}{d}}}{\triangle r}).$$

From the above time complexity, we can find that theoretically, the time cost of searching top-$K$ neighbors should be $k$ times that of top-1. However, in practice, we found that the time cost of searching top-$k$ neighbors by algorithms such as HNSW and NSG is far less than $k$ times that of top-1. Meanwhile, even if there exists $k$ monotonic search path from the query to its $k$ nearest neighbor. It is still hard for Algorithm 1 to strictly follow the search path since the top $k$ nearest neighbor can not be identified before reached. However, we found that the existing graph index such as HNSW and NSG will reversely connect its neighbor to the current point (reverse edge) after edge occlusion.

Then, we reconsider the property of MSNET. From the definition of MSNET, the $v_k$ is either connected to $q$ or a monotonic search path $v_k, v', ..., q$ exists and $||v_k, q|| > ||v', q||$ Because $v_k$ is the top $k$ nearest neighbor of $q$, then $v'$ must be one of $v_1, ..., v_{k-1}$ which is already visited. However, the edge from $v_k$ to $v'$ is a directed edge. Even though we have visited $v'$, we still cannot directly access $v_k$. After adding the reverse edges of all outgoing edges of $v_k$, we can directly access $v_k$ through the reverse edge from $v_k$ to $v'$. (Adding additional edges does not affect the property of MSNET, that is, the monotonic search path between two points still exists.)

From Algorithm 1 the neighbor of visited points are pushed into the candidate queue $C$. If the top $1, ..., k-1$ nearest neighbor is found, then $v_k$ will be at the top of the candidate queue and only 1 extra step is needed. Then we can derive that the expected search path length of MSNET for the top $k$ nearest neighbor is:

$$O(\frac{N^{\frac{1}{d}} \log N^{\frac{1}{d}}}{\triangle r} + k)$$

where only an extra $k$ step is needed.

**Proof of Theorem 2.** With the conclusion in Lemma 1 and Theorem 1, the expected search path of an MSNET with reverse edge finds the range-filtering nearest neighbor can be regarded as the expected path length that accumulates *in-range* $k$ nearest neighbor. Note that we only consider cases where $k$ is less than the range filter size. The RFKNN only exists in such a setting. Specifically, we regard the top $h$ nearest neighbor search as a drawing problem without replacement. Starting from the search for the top $k$ nearest neighbor, each search for the $k$+1th nearest neighbor can be regarded as drawing one from $N$ data points, where the *in-range* point is considered a success and the out-range point is considered a failure. In this way, the extra steps of implementing the top-$k$ range filtering nearest neighbor search through the top-$h$ nearest neighbor search can be regarded as the expected number of drawing times to accumulate a total of $k$ *in-range* points.

Formally, we set $N$ as the overall number of points, $K$ as the *in-range* number of points, and $L$ as the *out-range*. Our target is to find the expected number of draws $E[T]$ when the $k$-th *in-range* point is drawn. To find $E[T]$, we can use the method of order statistics, that is, consider the expected position of the $k$-th *in-range* in the number of draws among all possible arrangements of the index visited points.

With the assumption in [11] and the attribute values of points in $\mathcal{D}$ are independent. That is, for any $i \in [1, N]$, each point in $\mathcal{D}$ appears in the $i$-th position of the ordering of $\mathcal{D}$ with an equal probability. Then, assume that all $N$ points are randomly arranged with equal probability. We are interested in the position of the range-filtered $k$ nearest neighbors in this arrangement(sorted by distance to query $q$). Let the position of $k$ range-filtered nearest neighbor be $W_1 < W_2 < W_3, ..., < W_k$. Our goal is to find the expectation of $W_k$ the $E[W_k]$.

In the case of discrete uniform distribution, the expectation of the order statistic of the sequence position can be calculated by combinatorial methods. We first consider the total number of permutations. Select $K$ positions from $N$ positions to place the *in-range* point. There are $\binom{N}{K}$ options. For the number of cases where the $k$-th *in-range* point ($k$-th range filtering nearest neighbor) is at position $i$: There must be $k$-1 in-range points in the first $i$-1 positions, position $i$ is the $k$-th *in-range* point, and there are $K$-$k$ *in-range* points in the remaining positions $N - i$. Therefore, the number of permutations that meet the conditions is $\binom{i-1}{k-1}\binom{N-i}{K-k}$.

According to the definition of expectation:

$$E[W_k] = \sum_{i-k}^{N-K+k} i \times \frac{\binom{i-1}{k-1}\binom{N-i}{K-k}}{\binom{N}{K}}$$

Note that an important identity for the number of combinations is:

$$i\binom{i-1}{k-1} = k\binom{i}{k}$$

Substituting the above identity into the original expression we get:

$$\begin{aligned} E[W_k] &= \frac{1}{\binom{N}{K}} \sum_{i=k}^{N-K+k} i\binom{i-1}{k-1}\binom{N-i}{K-k} \\ &= \frac{1}{\binom{N}{K}} \sum_{i=k}^{N-K+k} k\binom{i}{k}\binom{N-i}{K-k} \\ &= \frac{k}{\binom{N}{K}} \sum_{i=k}^{N-K+k} \binom{i}{k}\binom{N-i}{K-k} \end{aligned}$$

We next calculate the summation term. Note the following combinatorial identity (a variation of the Vandermonde's identity)

$$S = \sum_{i=0}^{N} \binom{i}{k}\binom{N-i}{K-k} = \binom{N+1}{K+1}$$

Since in our summation, the number of combinations is zero when $i < k$ or $i > N-(K-k)$, the upper and lower limits of the summation can be extended to $i = 0$ to $N$ without affecting the result. Therefore, we have:

$$\begin{aligned} E[W_k] &= \frac{k}{\binom{N}{K}}\binom{N+1}{K+1} \\ &= k \times \frac{N+1}{K+1} \end{aligned}$$

Under the condition of satisfying elastic factor constraint, that is, $K \geq N \times c$, we further simplify the formula:

$$E[W_k] \leq k \times \frac{N+1}{N \times c + 1} = O(k/c)$$

So far, we have proved that the expected number of additional steps to search for the Top $k$ range filtering nearest neighbor is $O(k/c)$. We take $N' = |[L, R]|$, then the final expected search length can be written as:

$$O(\frac{N'^{\frac{1}{d}} \log N'^{\frac{1}{d}}}{\triangle r'} + k/c)$$

where $O(k)$ for extra step for top $k$ RFKNN search.

**Proof of Lemma 2.** We first find the corresponding segment tree node according to the segment tree split method. That is, if the ranges corresponding to the query and the right subtree overlap, then we directly access the right subtree, and the same goes for the left subtree. If the query range overlaps with the range of left and right subtrees of the current tree node, we first determine whether the current node satisfies the elastic factor constraint. If so, we use the graph index corresponding to the current node to perform the PostFiltering algorithm, that is, only one index is used. If the query does not satisfy the elastic factor constraint, we split the range filter $[l, r]$ of the query into $[l, mid]$ and $[mid, r]$ respectively. After splitting, we can convert the general query into two half-bounded queries with range filter $[l, mid]$ and $[mid, r]$. Let $[L, R]$ be the current segment tree range, and $[l, mid] \subseteq [L, mid]$ and $[mid, r] \subseteq [mid, R]$. For the query with filter $[l, mid]$. We only traverse the right subtree of the current node and the next node and determine whether it meets the elastic factor constraint. Each time we traverse the right subtree, the overall interval will be shortened by half, while the right bound remains unchanged. This search strategy is the same as the $\mathsf{ESG}_{1\mathrm{D}}$ index, and only one index is needed in the end. Similarly, the range filter $[mid, r]$ also only needs one index. Then we can prove that the $\mathsf{ESG}_{2\mathrm{D}}$ index uses two indexes for general queries in the worst case.

**Proof of Lemma 3.** Let $f$ denote the fanout of the segment tree, the elastic factor constraint is $1/f$. If the query range $[l, r]$ requires more than or equal to 3 subtrees to cover, then the middle index range will be completely covered by the query range. Let $[l_m, r_m]$ be one of the middle index ranges and $[l_r, r_f]$ be the father node index range. The elastic factor of $e([l_m, r_m], [l_m, r_m]$ will be equal to $1/f$. Since, $[l_q, r_q]$ cover $[l_m, r_m]$ and be covered by $[l_f, r_f]$, the elastic factor will greater than $1/f$. Then the father range index $[l_f, r_f]$ can be invoked for RFANN search which only uses one index. If the query range $[l, r]$ requires 2 subtrees to cover, we use the same split strategy to convert the general RFAKNN query into two half-bounded queries. Then only two indexes are used. If the query range can be covered by only one subtree, we follow the segment tree search strategy to keep searching.

## A.2 Discussion

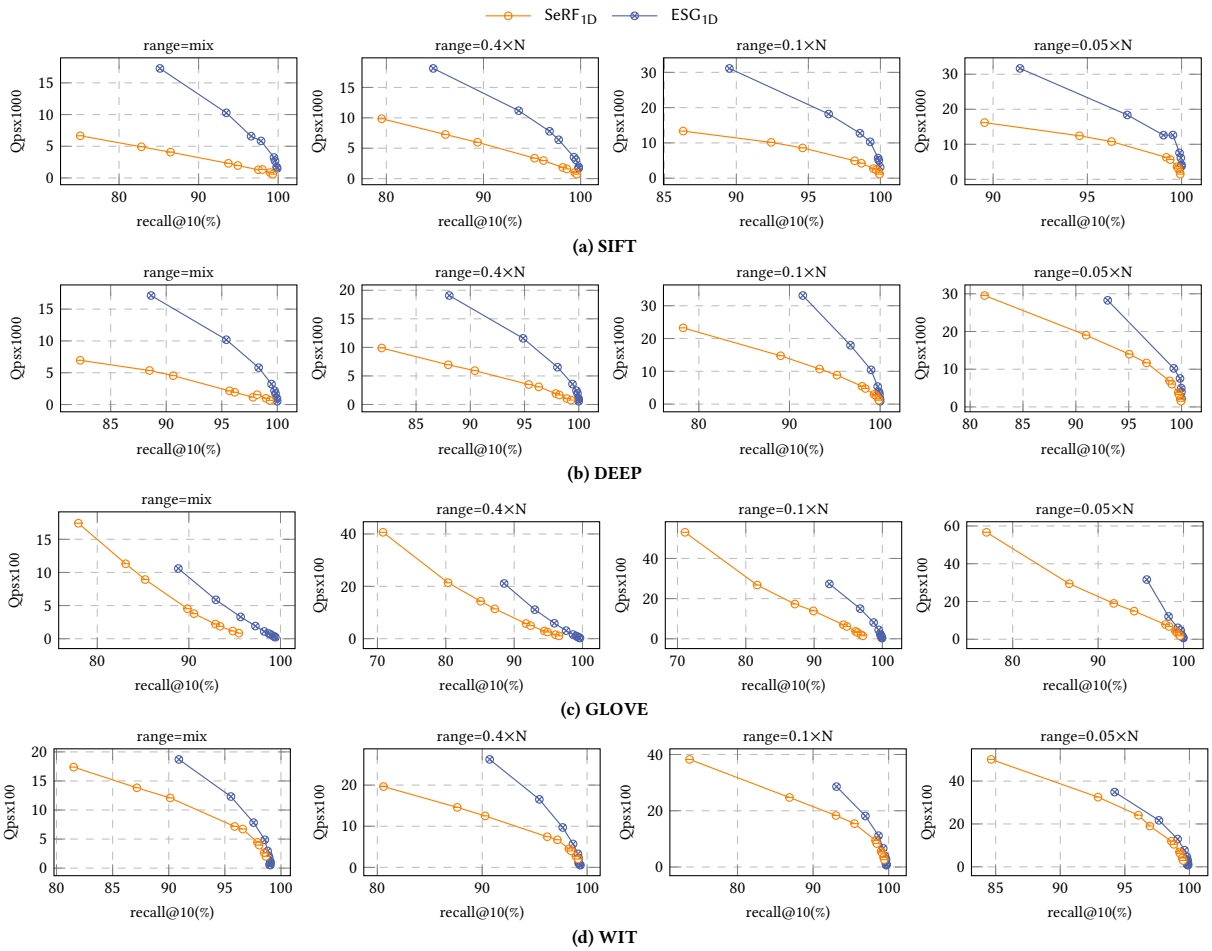
**Discussion of Modular Graph Index.** Our algorithms, $\mathsf{ESG}_{1\mathrm{D}}$, and $\mathsf{ESG}_{2\mathrm{D}}$, serve as frameworks with the default graph indexing algorithm set to HNSW, due to its widespread application and robust performance. It is worth noting that replacing the HNSW algorithm with more advanced algorithms like $\tau$-MG [35] or VAMANA [24] might improve time complexity or practice search efficiency. However, while $\tau$-MG offers better search time complexity, it incurs a larger out-degree for each node at $O(\log N)$, which leads to a higher spatial cost for the $O(\log N)$ layers required by $\mathsf{ESG}_{2\mathrm{D}}$. Meanwhile, HNSW supports continuous insertions, and we have significantly optimized index construct efficiency based on this feature.

Another solution to the problem of memory pressure caused by a large graph index is the disk AKNN search solution [24, 41]. In the graph-based disk search scenario, only the compressed vectors are stored in memory, and the full-precision vectors and graph indexes are stored on disk. For the $\mathsf{ESG}_{2\mathrm{D}}$ algorithm that requires a large memory space for storage, the graph structure is placed on the disk as the main space cost, so the memory cost is the same as the original disk-based graph-index methods that is, only the compressed vector is in memory.

## A.3 Additional Experimental Results

**Exp-A.1. Test of Varying Range Lengths on Half-Bound Queries.** We provide a variety of half-bounded range filter search performance tests in Fig. 12. Note that the use of a range filter from $[1, N/2^i]$ will lead to the $\mathsf{ESG}_{1\mathrm{D}}$ algorithm having the theoretical optimal performance since the elastic factor will be 1. From the experiment result, our algorithm has a stable performance improvement relative to the SeRF algorithm cross various range filters.

**Exp-A.2. Additional Experiments on Large-Scale Data.** For large-scale data, the indexing time and index space are typically large. A simple method is to use a segment tree with a larger fanout to reduce the number of index layers. The consequence of a larger



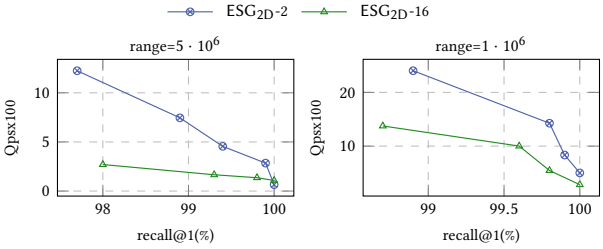

(a) SIFT

(b) DEEP

(c) GLOVE

(d) WIT

**Figure 12: The Test of Varying Range Lengths on Processing Half-Bound RFAKNN Queries**




(a) DEEP100M (General)

**Figure 13: The Large Scale Range Filter with Large Fanout**

fanout is a reduction in search performance. Therefore, we conducted additional experiments. As shown in Fig. 13, we compared the cases of fanout set as 2 ($ESG_{2D}$-2), and fanout set as 16 ($ESG_{2D}$-16). It can be seen that both settings can achieve a high search recall. Also, the performance gap between the two fanout settings is not significant at a high recall level. However, at a lower recall level, $ESG_{2D}$-2 has a 2x-6x performance advantage over $ESG_{2D}$-16. However, $ESG_{2D}$-2 requires 2x the build time and 3x the time cost of $ESG_{2D}$-16. Therefore, for a scenario with limited index memory space and build time, a $ESG_{2D}$ with a large fanout, such as 16, can be adapted.