Attribute Filtering in Approximate Nearest Neighbor Search: An In-depth Experimental Study

Mocheng Li
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
mochengli1@link.cuhk.edu.cn

Xiao Yan
Wuhan University
Wuhan, China
yanxiaosunny@gmail.com

Baotong Lu
Microsoft Research
Beijing, China
baotonglu@microsoft.com

Yue Zhang
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
223040247@link.cuhk.edu.cn

James Cheng
The Chinese University of Hong Kong
Hong Kong, China
jcheng@cse.cuhk.edu.hk

Chenhao Ma*
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
machenhao@cuhk.edu.cn

Abstract

With the growing integration of structured and unstructured data, new methods have emerged for performing similarity searches on vectors while honoring structured attribute constraints, i.e., a process known as Filtering Approximate Nearest Neighbor (Filtering ANN) search. Since many of these algorithms have only appeared in recent years and are designed to work with a variety of base indexing methods and filtering strategies, there is a pressing need for a unified analysis that identifies their core techniques and enables meaningful comparisons.

In this work, we present a unified Filtering ANN search interface that encompasses the latest algorithms and evaluate them extensively from multiple perspectives. First, we propose a comprehensive taxonomy of existing Filtering ANN algorithms based on attribute types and filtering strategies. Next, we analyze their key components, i.e., index structures, pruning strategies, and entry point selection, to elucidate design differences and tradeoffs. We then conduct a broad experimental evaluation on 10 algorithms and 12 methods across 4 datasets (each with up to 10 million items), incorporating both synthetic and real attributes and covering selectivity levels from 0.1% to 100%. Finally, an in-depth component analysis reveals the influence of pruning, entry point selection, and edge filtering costs on overall performance. Based on our findings, we summarize the strengths and limitations of each approach, provide practical guidelines for selecting appropriate methods, and suggest promising directions for future research. Our code is available at: https://github.com/lmccccc/FANNBench.

*Corresponding author.

SIGMOD 26, June 03–05, 2026, Bangalore, India

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts

• **Information systems** → **Retrieval efficiency**.

Keywords

Approximate Nearest Neighbor, Filtering, Benchmark, Survey

ACM Reference Format:
Mocheng Li, Xiao Yan, Baotong Lu, Yue Zhang, James Cheng, and Chenhao Ma. 2018. Attribute Filtering in Approximate Nearest Neighbor Search: An In-depth Experimental Study. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD 26).* ACM, Bangalore, India, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The advent of Transformer architectures [57] and modern embedding techniques [53] has led to the widespread use of vector representations for unstructured data such as text, video, audio, and images. These breakthroughs in representation learning have driven rapid progress in **Approximate Nearest Neighbor (ANN)** search technologies, whose core objective is to efficiently retrieve the top-k vectors that are approximately most similar to a given query vector (Figure 1a). ANN search is integral to vector databases [2, 4, 7, 44, 58, 61, 65], serving as the fundamental system for recommendation systems [43, 46, 51], pattern matching [14, 34], and retrieval-augmented generation (RAG) [20, 28, 29].

Various indexing methods have been developed for efficient ANN search, including graph-based [31, 59], quantization-based [24, 32], hashing-based [16, 27], and tree-based [10, 55] approaches. According to recent academic and industrial studies [11, 54, 60, 63], graph-based and quantization-based methods are most prominent. Graph-based algorithms construct neighborhood graphs that offer high query efficiency and recall by traversing from an entry point to successively closer neighbors [12, 35, 37, 49, 56]. On the other hand, quantization-based methods compress vectors—via dimensionality reduction [30] or lower-bit encoding [45, 48]—resulting in low memory usage, good GPU parallelism compatibility, and often integrate inverted file structures for enhanced efficiency.

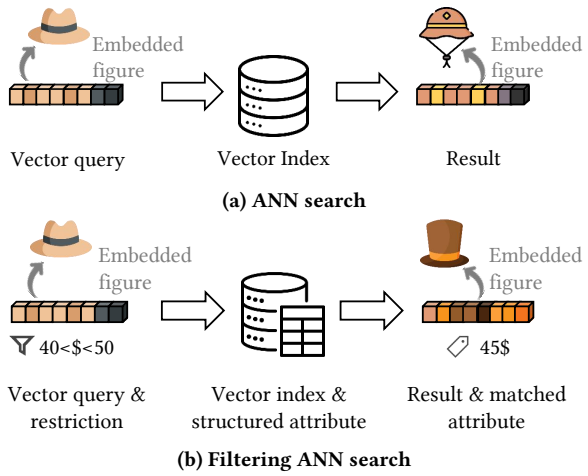


Figure 1: Example of ANN search and Filtering ANN search.

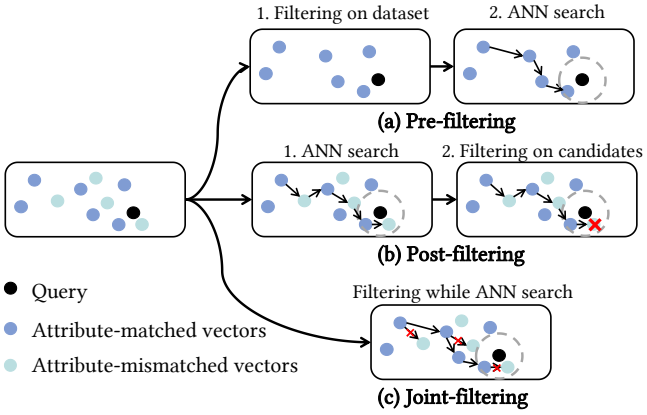


Figure 2: ANN filtering strategies.

Table 1: Attribute types and corresponding filtering types.

Attribute Type	Filtering Type	Query Predicates
Numerical	Range	$l \le O_i.a \le u$
Categorical	Label	$f \in O_i.A$
Arbitrary	General	Any filtering expression

1.1 Filtering ANN Search

Building upon the foundation of ANN search, there is a growing need for hybrid ANN search scenarios that integrate vector similarity search with relational database-style functionalities. This hybrid search treats vectors as a native data type while enabling SQL-like queries over structured attributes. For example, one might search for visually similar products while restricting results by price (see Figure 1b) or retrieve similar photos taken by the same individual. We refer to this hybrid search task as **Filtering Approximate Nearest Neighbor (Filtering ANN)** search.

Various Filtering ANN search methods have been developed through multiple integration strategies between attribute filtering and vector similarity search [6, 8, 42]. Moreover, some methods focus on specific attribute types for better performance [26, 59, 67], but this complicates the classification of filtering tasks. To effectively handle different attribute types, we provide a systematic taxonomy from two perspectives: attribute types and filtering strategies.

Filtering Attributes. Current systems or methods predominantly target two attribute types: *Numerical* and *Categorical*, corresponding to *Range Filtering ANN search* and *Label Filtering ANN search*, respectively.

- *Numerical Attributes and Range Filtering.* Numerical attributes, such as price or date, represent continuous values. Range filtering retrieves nearest neighbors whose attribute values fall within a specified interval (e.g., products within a price range or photos taken between two dates). For a range Filtering ANN query, let O_i denote a vector–attribute pair (v, a) with index i, where $O_i.a$ represents its numerical attribute. Let l and u denote the query's lower and upper bounds, respectively, as shown in Table 1. Several approaches, including SeRF [67], iRangeGraph [62], and UNIFY [38], have been designed to address this scenario.
- *Categorical Attributes and Label Filtering.* Categorical attributes represent discrete labels. For example, YouTube videos often carry tags like "music", "comedy", or "news" that describe their content. Label filtering finds nearest neighbors (i.e., similar videos) that share one or more of these tags. In Table 1, f denotes the query's attribute, while $O_i.A$ represents the categorical attribute set of O_i. Filtered DiskANN [26] is a classic example of this approach.
- *Arbitrary Filtering.* Beyond categorical and numerical filters, some systems support more flexible filtering restrictions via a user-defined filtering function. These methods either integrate closely with traditional databases, e.g., VBASE [65] and Milvus [58] or manage arbitrary subset searches while abstracting away filtering details, e.g., Faiss [19] and ACORN [50].

Filtering Strategies. Figure 2 illustrates common approaches to integrating attribute filtering with ANN search. These approaches differ in the order in which filtering is applied relative to the ANN search process, and we classify them into three categories: *pre-filtering*, *post-filtering*, and *joint-filtering*. In this context, *selectivity* refers to the fraction of data items that pass a filtering condition: high selectivity means many pass, while low selectivity means few do.

- *Pre-filtering* applies filtering prior to the similarity search to reduce the search space—effective for data-independent indexes such as the Inverted File Index (IVF). However, in graph-based methods, pruning nodes based on attributes may disrupt traversal paths and reduce recall.
- *Post-filtering* performs the ANN search first and then removes candidates that do not meet the attribute constraints. This approach is efficient when filter selectivity is high but becomes costly in low-selectivity scenarios, as many invalid candidates need to be discarded after the search.
- *Joint-filtering* integrates attribute filtering directly into the ANN search. For example, in a graph-based search, the traversal is restricted to edges connecting to nodes that satisfy the attribute

constraints, allowing the algorithm to prune paths early. This dynamic approach is especially effective when filter selectivity is moderate.

1.2 Our Contributions

Motivated by the rising demand for efficient Filtering ANN algorithms and the diversity of existing filtering strategies, we conduct a comprehensive experimental survey to clarify their strengths, limitations, practical application scenarios, and open problems. Our contributions are summarized as follows:

Taxonomy Study (Section 3). While numerous methods have reported impressive performance gains, it remains challenging to identify their core ideas, advantages, and limitations. We systematically review 12 systems and algorithms, providing a unified classification based on their indexing methods, filtering strategies, and key techniques (Figure 4 and Table 3). Our two-axis taxonomy categorizes methods by filtering type (label, range, and arbitrary filtering) and by filtering strategy (pre-filtering, post-filtering, and joint-filtering), offering clear insights into their design trade-offs.

Comprehensive Experimental Evaluation (Section 5.2, 5.3 and 5.5). Given that most algorithms in this area have been published recently and involve numerous hyperparameters, comparisons remain challenging. To address this, we develop a unified interface that enables consistent evaluation, and we conduct extensive experiments across 12 methods from 10 representative papers, covering both label and range filtering tasks with query selectivity ranging from 0.1% to 100%. Our results indicate that segmented graph indexing excels in range filtering scenarios, while label filtering techniques are often unstable due to suboptimal graph quality.

Component Analysis (Section 4, 5.6, 5.7, and 5.8). We analyze the core components of Filtering ANN algorithms to determine which factors contribute most significantly to overall performance. Our analysis reveals that constructing an index for all possible subsets is both intuitive and effective. Additionally, we identify underexplored components in graph-based methods, such as pruning strategies and entry point selection, that are crucial for performance. We have several interesting findings: (1) Relative Neighbor Graph (RNG) pruning breaks down at low selectivity; (2) hierarchical multi-layer indexes boost performance only at high selectivity and are otherwise optional; and (3) bypassing hierarchy in these indexes by increasing bottom-layer entry points consistently improves performance across all selectivity levels.

Usage and Development Guidelines (Section 6). Recognizing that Filtering ANN algorithms are designed for distinct scenarios, we propose practical guidelines (Figure 15) to help practitioners select the most suitable method based on specific filtering needs and data characteristics. We also discuss emerging trends and potential future research directions to drive further innovations in this field.

2 Preliminaries

Most Filtering ANN algorithms build upon Inverted File (IVF) or graph-based methods, often enhanced with quantization strategies. To set the stage for filtering methods, we briefly review these foundational ANN approaches, with key notations summarized in Table 2.

Table 2: Notations in our paper.

Notations		Descriptions
$\mathcal{D} = \{O\}$		Vector dataset with attributes.
O	$= (v, a)$	Vector and numerical attribute.
	$= (v, A)$	Vector and categorical attribute set.
$Q = (q, r)$		Label query vector and its restriction.
r	$= (l, u)$	Lower/upper bound for range query.
	$= f$	Label for label query.
n		Size of $\mathcal{D}$.
$\mathcal{D}_r$		Subset of $\mathcal{D}$ filtered by the restriction r.
d		Dimension per vector in V.
M		Maximum degree for graph ANN index.
$\phi(v_i, v_j)$		Similarity between vector v_i and v_j.

(a) Edge not to be pruned. **(b) Edge to be pruned.**

Figure 3: RNG pruning example.

2.1 Vector Quantization

Efficient similarity search on high-dimensional data often requires reducing the computational burden. One common approach is to reduce the bit size of each dimension through quantization. For example, Product Quantization (PQ) [32, 48] compresses vectors by clustering one or more dimensions, typically represented as 32-bit floating-point values, into 256 clusters. Each cluster is encoded with an 8-bit code, approximating the original values with substantially lower memory cost.

More advanced quantization techniques, such as Optimized Product Quantization (OPQ) [25], Scalar Quantization (SQ) [15], and RabitQ [24], build on these principles to improve encoding accuracy and computational efficiency, demonstrating strong performance in practical applications.

2.2 Inverted File

Another key building block is the Inverted File (IVF) method, which divides vectors into several partitions using k-means clustering [39]. During a search, only the partitions closest to the query vector are processed.

IVF is commonly combined with quantization techniques (e.g., PQ), forming systems such as IVFPQ [19, 32], to further decrease both computation and memory overhead.

2.3 Graph-Based ANN Search

Graph-based ANN search builds an index by connecting vectors based on distance. Key differences among these methods stem from their *pruning strategies* and *entry point* selection. While many approaches exist [60], only a few are suitable for Filtering ANN. Below, we summarize the most widely used graph indexes in this context.

KGraph [18] initially connects vectors randomly. For each vector, it iteratively refines the neighbors to the nearest ones by examining the neighbors of its neighbors, following the rule that *neighbors are more likely to be neighbors of each other* [22]. This design allows KGraph to be constructed efficiently, but it sacrifices connectivity, often resulting in multiple disconnected components [60].

Relative Neighbor Graph (RNG) [56] does not initialize a random graph but instead searches for the nearest neighbors of each vector to be inserted and connects them iteratively. The connection step employs a pruning-based strategy to build a high-quality graph, taking into account the spatial distribution of neighbors. For instance, in Figure 3, suppose point v_1 has candidate neighbors v_2 and v_3, and edge $e(v_1, v_2)$ already exists. RNG prunes edge $e(v_1, v_3)$ if v_3 is closer to v_2 than to v_1, implying that a direct connection is redundant. This strategy maintains a well-connected and scalable structure while limiting the number of neighbors per node. Modern graph-based ANN indexes, such as Vamana Graph [31], NSG [23], NSW [40], and HNSW [41], are built upon RNG or its variants.

Vamana Graph (VG) [31], introduced in DiskANN, starts from a randomly connected graph, performs a one-pass ANN search for each vector, and then prunes each vector's neighbor list to at most M using the pruning strategy of RNG. This design also retains edges to remote vectors, thereby ensuring connectivity and accelerating search convergence.

Hierarchical Navigable Small World (HNSW) [41] constructs a multi-layer graph index where each higher layer is a subsampled version of the one below. In each layer, HNSW leverages the Navigable Small World (NSW) principle by preserving both nearest neighbors for local connectivity and long-range links for global navigation. It also employs an RNG-inspired pruning strategy to eliminate redundant edges, thus ensuring the graph remains sparse yet well-connected. With a single entry point at the top layer, this hybrid design guarantees efficient index construction and rapid query processing.

2.4 Filtering ANN Search

Formally, the *Filtering Nearest Neighbor Search (Filtering NN)* is defined as:

$$NN(q|r) = \arg \min_{o \in D, r(o.a)} ||q - o.v||,$$

where $NN(q|r)$ denotes the exact nearest neighbor whose vector $o.v$ is closest to the query vector q under a given distance metric (e.g., L2, cosine, or inner product), and whose attribute $o.a$ satisfies predicate r.

In practice, retrieving exact results is often prohibitive, so approximate solutions (*Filtering ANN*) are used as the algorithmic search target to improve efficiency.

3 Overview of Filtering ANN Algorithms

Building on the foundational techniques discussed earlier, recent research has advanced ANN search in filtering scenarios by developing diverse algorithms that combine vector similarity with attribute filtering. Table 3 presents a selection of these methods, which integrate various structural designs and core ideas to achieve distinct performance. In the following subsections, we detail approaches for range, label, and arbitrary filtering. Figure 4 illustrates our taxonomy of the algorithms and their relationship to key techniques.

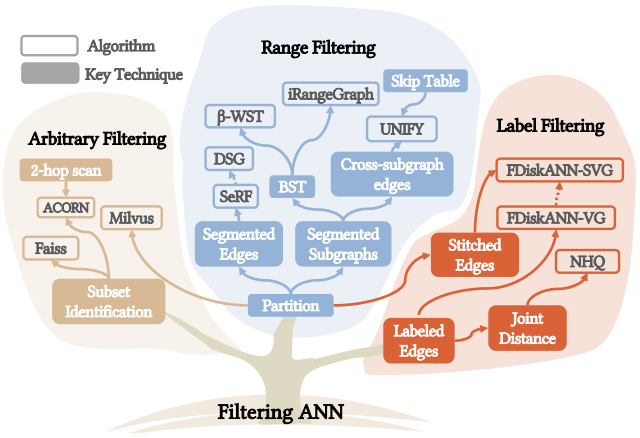


Figure 4: A roadmap of Filtering ANN algorithms.

Index partitioning is employed across all filtering methods, enabling the search range to be pre-defined during construction, especially for range filtering. Label filtering tags edges or encodes labels in distances, while arbitrary filtering identifies subsets before search.

3.1 Range Filtering ANN Search

Range Filtering ANN search exploits the natural ordering of continuous numerical attributes, inspiring a diverse set of algorithmic approaches.

SeRF [67] constructs range-aware edges by first sorting the dataset by attribute values and then incrementally inserting vectors into the graph in ascending order. For each vector v_i, SeRF performs neighbor searches over all subranges $[j, i]$ (with $j \leq i$), by creating *segmented edges*. For example, it first identifies the top M neighbors for the range $[0, i]$, then for $[1, i]$, $[2, i]$, and so on. This segmentation implicitly encodes range constraints via insertion order. During search, SeRF abandons the traditional HNSW structure and instead selects three entry points within the queried range, which enhances efficiency by guiding the search only through valid edges.

DSG [52] builds on SeRF by optimizing construction efficiency. Its key innovation is dynamic vector insertion, which eliminates the need for pre-sorting, thereby enabling real-time index updates.

β-WST [21] tackles range filtering by constructing a binary search tree (BST) based on attribute ranges, with each BST node hosting a dedicated subgraph that contains only the vectors within that node's attribute range. This hierarchical design yields $\log(n)$ layers, confining the search to the subgraphs relevant to the query range. In its basic form, a query may intersect multiple subgraphs, potentially reducing efficiency. To address this, an enhanced variant, OptPostFiltering, introduces controlled subgraph overlap combined with post-filtering to streamline the search process.

iRangeGraph [62] also leverages a BST to build $\log(n)$ layers of subgraphs but differs in query execution. Instead of independently searching subgraphs and merging results, iRangeGraph collects

Table 3: Filtering ANN algorithms.

Algorithm	Filtering Type	ANN Index	Filtering Strategy	Attribute Index
SeRF [67]	Range	HNSW	Joint-filtering	Segmented edges
DSG [52]	Range	HNSW	Joint-filtering	Segmented edges
β-WST [21]	Range	VG	Pre-filtering	Segmented subgraphs (Binary search tree)
UNIFY [38]	Range	HNSW	Pre/Post/Joint-filtering	Segmented subgraphs (Cross-subgraph edges) + Skip list
iRangeGraph [62]	Range	RNG	Pre-filtering	Segmented subgraphs (Binary search tree)
FDiskANN-VG [26]	Label	VG	Joint-filtering	Labeled edges
FDiskANN-SVG [26]	Label	VG	Joint-filtering	Labeled edges + Stitched graph
NHQ [59]	Label	NSW/KGraph	None	Joint distance
Milvus [58]	Arbitrary	HNSW/IVF	Pre-filtering	Partition
Faiss-HNSW [19, 41]	Arbitrary	HNSW	Post-filtering	Subset identification
Faiss-IVFPQ [19, 32]	Arbitrary	IVF	Pre-filtering	Subset identification
ACORN [50]	Arbitrary	HNSW	Joint-filtering	Subset identification + Two-hop scan

entry points from all subgraphs matching the query range and uses them to traverse the subgraphs as if they are unified, thus avoiding the overhead of post-filtering or result merging.

UNIFY [38] partitions the dataset into attribute-based segments and builds subgraphs on a unified HNSW index. Subgraphs are linked via *Cross-subgraph edges*, with each edge annotated by a *mask list* for segment membership. The search strategy adapts dynamically based on selectivity and the hyperparameters `Sel_low` and `Sel_high`:

- Small-range queries (selectivity < `Sel_low`): Uses a *skip list* to pre-filter and scan only the matched vectors.
- Medium-range queries (selectivity `Sel_low`~`Sel_high`): Applies joint-filtering within the pertinent segmented subgraph.
- Large-range queries (selectivity > `Sel_high`): Switches to post-filtering over the full HNSW index.

Although coarse segmentation can be challenging for low-selectivity queries, UNIFY's adaptive strategy selection maintains efficient performance across diverse query types.

3.2 Label Filtering ANN Search

Label filtering retrieves similar vectors that satisfy categorical constraints in datasets with a limited number of distinct labels.

Filtered-DiskANN [26] extends the Vamana Graph (VG) [31] by introducing *labeled edges* to navigate query searching across valid nodes. Filtered-DiskANN introduces the *Stitched graph* to enhance connectivity, which builds subgraphs for individual categorical values and then merges them into a unified graph. Although this approach slows index construction, it significantly improves filtering performance.

NHQ [59] supports multi-attribute queries where each vector is labeled with exactly one value per attribute (e.g., color, trademark, and origin), and queries must specify a value for every attribute. To combine vector similarity with label matching, NHQ employs a *joint distance* metric defined as:

$$\phi'(O_i, O_j) = w_1 \cdot \phi(O_i.v, O_j.v) + w_2 \cdot \sum_{t=1}^{|O.A|} \mathbb{I}\{O_i.A_t = O_j.A_t\},$$

where $\phi(O_i.v, O_j.v)$ is the vector distance between O_i and O_j, $\mathbb{I}\{O_i.A_t = O_j.A_t\}$ is an indicator function that returns 1 if the t-th attribute of O_i matches that of O_j and 0 otherwise, and w_1 and w_2 balance the contributions from the vector and attribute similarities. Although NHQ efficiently guides search using this heuristic, its results may not always perfectly match the filter criteria.

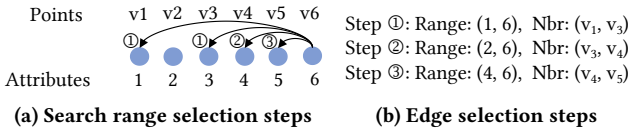
3.3 Arbitrary Filtering ANN Search

Arbitrary filtering encompasses scenarios where filtering conditions are flexible, allowing users to define custom subsets of the dataset or custom filtering functions.

Faiss [19] is a library that supports various ANN methods (e.g., HNSW, LSH, IVF) as well as quantization techniques (e.g., PQ, SQ). It enables arbitrary filtering via *Subset identification*, which labels matched items before Filtering ANN search. The `is_member(id)` function checks whether a vector belongs to the subset that meets the filtering constraint during search. In Faiss's HNSW implementation, `is_member(id)` is invoked during the final layer search to filter results (post-filtering), whereas in IVFPQ it is applied after selecting the `nprobe` clusters and before computing distances (pre-filtering), thus avoiding unnecessary computations.

Milvus [58], built on Faiss, is a vector database that supports arbitrary filtering conditions. Milvus is popular for being a representative of several vector database systems for its high efficiency, versatility [47], and widespread adoption in LLM applications [20]. It improves filtering performance by first *partitioning* the dataset into multiple subsets (64 by default) based on a specified attribute, which simplifies search tasks confined to specific segments.

ACORN [50] employs a predicate-agnostic strategy by ignoring filtering constraints during index construction. Its indexing method is similar to HNSW but incorporates two key modifications: it prunes neighbors using a *two-hop* rule (see Section 4.2), and, during the search, it expands to two-hop neighbors when immediate neighbors are insufficient. While effective for arbitrary filtering, this general-purpose connectivity may lead to slower convergence for categorical or numerical queries compared to specialized methods.



(a) Search range selection steps **(b) Edge selection steps**

Figure 5: Segmented edge selection example. M = 2; the order of distances to v_6 is: v_1, v_3, v_4, v_5, v_2; 'Nbr' denotes the selected neighbors for each range.

4 Detailed Analysis of Key Components

In this section, we examine the core techniques and design components of Filtering ANN algorithms, emphasizing their distinctive features and implementation nuances.

4.1 Attribute Index

We focus on analyzing range Filtering ANN indexes, as they are significantly more complex and worthy of detailed examination. Range filtering relies on partitioning the dataset according to the natural ordering of numerical attributes. As used in Milvus, a simple strategy is to divide the dataset into segments and build a separate index for each subset. However, many range filtering algorithms employ more sophisticated segmentation strategies. Based on the techniques summarized in Table 3 and Figure 4, we classify them into two main approaches: **segmented edges** and **segmented subgraphs**.

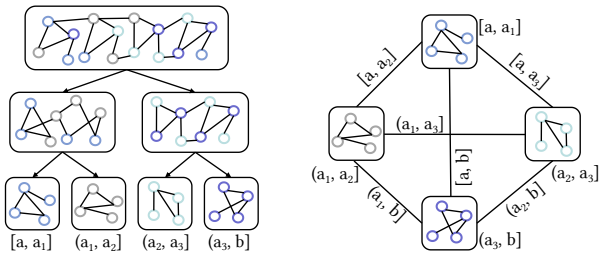
Segmented edges. SeRF and DSG build edges annotated with range indicators that support queries over different intervals, ensuring that every query can effectively locate its target vectors. The core idea is to create edges covering all possible query ranges. Figure 5a illustrates how the new node v_6 is inserted and connected to existing nodes, while Figure 5b presents its edge sets along with their corresponding filtering ranges. The algorithm performs a progressive search, from the largest to the smallest query range, as follows:

①. For range $(1, 6)$, select the nearest neighbors v_1 and v_3.
②. For range $(2, 6)$, choose $\{v_3, v_4\}$, reusing distances computed in ①.
③. Skip range $(3, 6)$ as it shares the same neighbors as $(2, 6)$; for range $(4, 6)$, select v_4 and v_5.

Technically, DSG constructs one extra range for each edge, ensuring that they can guide range queries that do not match the current node and help find closer paths. Additionally, incremental insertion is supported, as the insertion order is no longer a strict requirement.

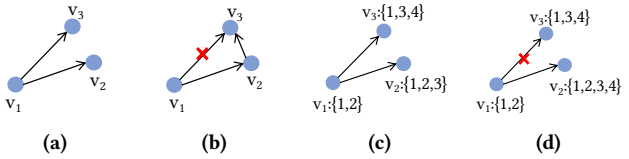
Segmented subgraphs. Another strategy for range Filtering ANN indexing is to build dedicated subgraphs for different query intervals. For instance, β-WST and iRangeGraph organize data into a BST, with each tree node hosting a subgraph that covers a specific range, as illustrated in Figure 6a. Queries then combine the relevant subgraphs from the corresponding BST layers. UNIFY adopts a similar approach but creates a fixed set of disjoint subgraphs—one per range—and introduces Cross-subgraph edges to maintain global connectivity, as shown in Figure 6b.

A BST can produce up to $\log(n)$ layers, resulting in $2n - 1$ subgraphs and supporting a large spectrum of range combinations.



(a) Binary search tree **(b) Cross-subgraph edges**

Figure 6: Different implementations of segmented subgraph approaches.



(a) **(b)** **(c)** **(d)**

Figure 7: Examples of two-hop and label pruning: (a) Edge not to be pruned in two-hop pruning, (b) Edge to be pruned in two-hop pruning, (c) Edge not to be pruned in label-covered pruning, and (d) Edge to be pruned in label-covered pruning.

By contrast, UNIFY's subgraph count remains fixed, generating fewer possible combinations. Consequently, UNIFY relies on post-filtering for specialized or narrow ranges, reducing its efficiency under low-selectivity conditions.

4.2 Pruning Techniques

Many Filtering ANN algorithms (e.g., Milvus, SeRF, DSG, β-WST, iRangeGraph, UNIFY) employ RNG pruning to boost search efficiency. For each vector, RNG pruning removes edges to vectors that are too close to its current neighbors, retaining only distinctive, diverse connections. However, since RNG pruning considers only distance and ignores attribute diversity, a crucial factor in filtering scenarios, several algorithms have introduced specialized pruning techniques.

Two-hop pruning. ACORN restricts pruning to *two-hop* neighbors (Figure 7b) without caring about relative distances like RNG but aligning with its two-hop *search* strategy. By excluding immediate neighbors from pruning, ACORN retains more potentially useful connections, improving both search efficiency and accuracy.

Label-covered pruning. Filtered-DiskANN enhances RNG pruning by integrating label information. An edge $e(v_1, v_3)$ is pruned only if two conditions hold: (1) the RNG condition is met—that is, there exists a vector v_2 such that $\phi(v_1, v_2) < \phi(v_1, v_3)$ and $\phi(v_3, v_2) < \phi(v_1, v_3)$; and (2) v_2's attribute set covers those of both v_1 and v_3. For instance, Figure 7d shows that $e(v_1, v_3)$ is pruned when both conditions are satisfied, whereas in Figure 7c only the RNG condition holds, so the edge is retained. This approach ensures that edges are pruned only when both spatial proximity and label consistency warrant it, thereby preserving crucial connectivity for categorical filtering tasks.

4.3 Entry Point Strategies

Graph-based Filtering ANN search must eliminate mismatched neighbors at various stages, and a critical challenge is selecting reliable entry points for graph traversal. Although Table 3 provides a general classification, important implementation nuances warrant further discussion.

Unrestricted entry points. Traditional approaches—such as Faiss-HNSW with `is_member(id)` restrictions—preserve the original entry points for queries. Similarly, post-filtering methods in ACORN, NHQ, and UNIFY initiate searches from default entry points of the graph index, gradually converging on matching results.

Specialized entry points. SeRF and DSG bypass the hierarchical navigation of HNSW by directly selecting multiple entry points from the bottom layer that satisfy the query range constraints. Instead of using a hierarchical scheme, they select entry points at evenly spaced intervals within the valid range, ensuring a well-distributed set of starting points without extra overhead.

Other methods handle entry point selection differently. For example, iRangeGraph dynamically combines entry points from multiple matched subgraphs during query processing, while UNIFY's joint-filtering uses a single top-level entry point from selected subgraphs and connects them via cross-subset edges. In its pre-filtering mode, UNIFY begins from a fixed entry point and delays distance computations until the skip table reaches the query's left bound.

5 Experiments

In this section, we present our experimental setup and results, offering a comprehensive evaluation of various Filtering ANN algorithms along with an in-depth analysis of their key components.

5.1 Setup

Platform. We conducted our experiments on Ubuntu 24.04 LTS, equipped with Intel® Xeon® Platinum 8358 CPUs @ 2.60GHz, x86-64 architecture, and 2TB of memory, with 128 physical cores. We use 128 threads for index construction. As query tasks are read-only, all methods exhibit similar performance trends with increased parallelism, so we use one thread during the query phase.

Metrics. All datasets use L2 (Euclidean) distance for similarity measurement. We evaluate algorithms using `Queries Per Second (QPS)`, where a higher QPS indicates faster processing, and `Comparisons` per query for graph-based algorithms, where a fewer comparisons signifies more efficient navigation. Since all methods apply similar SIMD-based techniques for distance computation, `QPS` and `comparisons` exhibit consistent trends.

Dataset. Our experiments are conducted on four datasets: SIFT [11, 33], SpaceV [13], Redcaps [17], and Youtube-RGB (see Table 4). These datasets include up to 10 million vectors with varying dimensions and label types (synthetic or real). Unless specified otherwise, real labels are used for Redcaps and Youtube-RGB.

To ensure a fair comparison across all algorithms, synthetic labels are generated using the following strategy: each vector is assigned an integer value between 0 and 100,000 for numerical attributes, and one of 500 integer values for categorical attributes, maintaining consistent label cardinality for both Filtered-DiskANN and NHQ. For each query task, we retrieve 10,000 items sequentially to evaluate the performance of all algorithms. For range queries, we control selectivity by adjusting the upper and lower bounds of the queried attribute. For label queries, we assign categorical attributes using fixed probabilities to control selectivity (e.g., assigning Label 1 to 50% of vectors so that querying with Label 1 yields 50% selectivity).

[1]http://corpus-texmex.irisa.fr/
[2]https://github.com/microsoft/SPTAG/tree/main/datasets/SPACEV1B
[3]https://redcaps.xyz/
[4]https://research.google.com/youtube8m/download.html

Table 4: Dataset statistics.

Dataset	Dim	Labels	Size	Query Size
SIFT [1]	128	Synthetic	10M	10K
SpaceV [2]	100	Synthetic	10M	10K
Redcaps [3]	512	Real/Synthetic	1M	10K
Youtube-RGB[4]	1024	Real/Synthetic	1M	10K

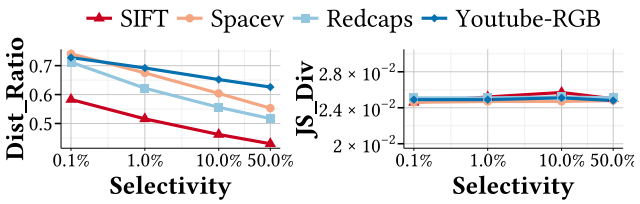


Figure 8: Query hardness of all datasets.

Figure 8 analyzes dataset hardness under varying selectivity levels. We define Dis_Ratio as the ratio between the average distance of the top-10 ground truth and the average pairwise distance in the dataset; a higher value implies a broader search range. We also estimate the Jensen-Shannon divergence (JS_Div) between the full dataset and the queried subset. In general, lower selectivity leads to harder queries, and both synthetic and real datasets exhibit limited divergence in query distribution.

Algorithms. In our experiments, we evaluate the latest Filtering ANN algorithms, grouped into three categories:

- **Range Filtering Methods:**
  - SeRF [67] and DSG [52]
  - β-WST [21], evaluated in two modes: the Vamana graph filtering method (WST-Vamana) and a super-optimized post-filtering variant (WST-opt)
  - iRangeGraph [62] and UNIFY [38], evaluated in two configurations: a hybrid pre-/post-/joint-filtering strategy (UNIFY-CBO) and a joint-filtering-only setup (UNIFY-joint)
- **Label Filtering Methods:**
  - Filtered-DiskANN [26], evaluated in both its base (FDiskANN-VG) and stitched (FDiskANN-SVG) forms
  - NHQ [59], implemented with NSW (NHQ-NSW) and KGraph (NHQ-KGraph)
- **Arbitrary Filtering Methods:**
  - Faiss [19] (Faiss-HNSW, Faiss-IVFPQ)
  - Milvus [58] (Milvus-HNSW, Milvus-IVFPQ)
  - ACORN [50]

To ensure fairness, we standardize key settings: parallelism is enabled for SeRF and DSG, ACORN's filtering storage is optimized, and Faiss's `is_member()` is improved. All methods use in-memory indexes, including DiskANN, which has a built-in in-memory search component. All graph-based methods use $M = 40$ and `ef_construction` = 1000. These settings yield near-optimal performance for up to 10M datasets [41, 52]. Each experiment is averaged over three runs. Full configurations are in the appendix [9].

5.2 Range Filtering: Performance and Analysis

We evaluate algorithm performance over selectivity levels of 0.1%, 1%, 10%, and 50% to assess efficiency under various query scenarios. Numerical attributes are generated based on the rules in Section 5.1. We fine-tune search hyper-parameters (e.g., `ef_search`, `nprobe`, etc.) to achieve optimal performance at 90% recall. Results are shown in Fig. 9. Missing entries indicate failure to meet the target recall under equivalent indexing configurations. Our key observations are as follows:

1. Partitioning ensures query availability at low selectivity. Milvus achieves reliable query service at 0.1% selectivity, a challenging regime where Faiss-HNSW fails completely. This capability stems from its partitioning mechanism: by distributing the dataset across smaller subsets based on attributes, the selectivity within each partition becomes significantly higher than the global selectivity.

2. IVF-based methods show reliable recall. Although Faiss-IVFPQ generally achieves lower QPS than Faiss-HNSW, it consistently handles 0.1% selectivity across all datasets. In contrast, Faiss-HNSW with post-filtering and ACORN often fail under these conditions. This discrepancy is due to the monotonic search behavior of HNSW, which restricts exploration as the search converges on the target, whereas the non-monotonic nature of IVF enables it to explore a broader search range and reliably locate matching targets.

3. The benefits of attribute-aware indexing are less pronounced at high selectivity levels. Our experiments reveal that when selectivity exceeds 50%, even traditional methods like Faiss-HNSW achieve competitive performance. This is expected, as the benefit of attribute-aware indexing is most pronounced under stringent filtering conditions; at high selectivity, the search range is broad enough that the extra filtering constraints provide little advantage.

4. Range filtering methods achieve similar performance in most cases. UNIFY-hybrid, WST-opt, iRangeGraph, SeRF and DSG consistently yield the highest QPS in most test scenarios (1% and 10% selectivity), thanks to subgraph constructions that precompute edges across all query ranges, ensuring robust connectivity during search. This demonstrates that segmented edges and subgraphs offer similar effectiveness for range filtering.

5. BST excels at low selectivity compared to cross-subgraph edges. iRangeGraph and WST-Vamana employ fine-grained subgraphs constructed using BST, achieving outstanding performance at 0.1% selectivity. In contrast, UNIFY-hybrid's default partitioning into 8 subsets performs well at higher selectivity levels but suffers from reduced QPS at 0.1%, even falling behind UNIFY-CBO's linear scan based on the skip table.

6. Segmented edges fail at low selectivity. Although SeRF and DSG deliver high QPS at high selectivity, both methods fail at 0.1% selectivity. Because they rely on a graph-based structure, this issue mirrors the monotonicity challenge found in Faiss-HNSW. Under extremely narrow filtering ranges, the initial search during index construction fails due to the absence of valid neighbors, leading subsequent queries to collapse. Notably, DSG underperforms compared to SeRF, primarily because it dedicates significant time to edge filtering despite its more refined strategy. Section 5.8 discusses this issue in detail.

5.3 Label Filtering: Performance and Analysis

Following the attribute generation method in Section 5.1 and experimental settings in Section 5.2, we conduct experiments for label filtering. Figure 10 presents the QPS achieved by each label filtering algorithm while maintaining 90% recall across various selectivity levels. Our analysis reveals several key insights:

1. Robustness of the stitched method. FDiskANN-SVG generally performs well across datasets and selectivity levels due to its stitched subgraph design. However, it fails to achieve 90% recall on SIFT, likely owing to limitations in the quality of the underlying Vamana Graph. In contrast, on SpaceV, Redcaps, and Youtube-RGB, FDiskANN-SVG meets or exceeds the recall target, indicating its effectiveness is dataset-dependent.

2. Limitations of joint distance at low selectivity. Both NHQ-KGraph and NHQ-NSW suffer substantial performance drops when selectivity falls below 1%. This decline stems from difficulties in maintaining graph connectivity under the joint distance metric; when nodes sharing the same label are sparse, the metric fails to establish valid neighbor connections, limiting its effectiveness for high-precision filtering.

3. Simpler pruning outperforms in label filtering search. Our comparison between NHQ-KGraph and NHQ-NSW reveals that simpler pruning strategies can yield superior performance in filtered search. By retaining all nearest neighbors, NHQ-KGraph preserves more connections, leading to improved recall. These findings suggest that, in Filtering ANN contexts, straightforward connectivity rules can outperform more complex geometric pruning techniques.

5.4 Arbitrary Filtering Analysis

To evaluate arbitrary filtering performance, we perform two-predicate searches using both categorical and numerical attributes for hybrid filtering. To achieve 50% overall selectivity, we assign the categorical label "1" to 60% of vectors and apply a range query with 83% selectivity. Under uniform distributions, the combined selectivity is approximately $60\% \times 83\% \approx 50\%$. Other selectivity levels are computed similarly. Since Milvus supports partitioning by only one attribute, we use the numerical attribute for partitioning. The results are presented in Figure 11.

1. Faiss-HNSW and ACORN show competitive performance at high selectivity. At selectivities greater than 10%, ACORN and Faiss-HNSW achieve higher QPS due to their single-index design without partitioning. Notably, Faiss-HNSW performs best when selectivity exceeds 50%. These results confirm that Faiss-HNSW remains a strong candidate for arbitrary filtering.

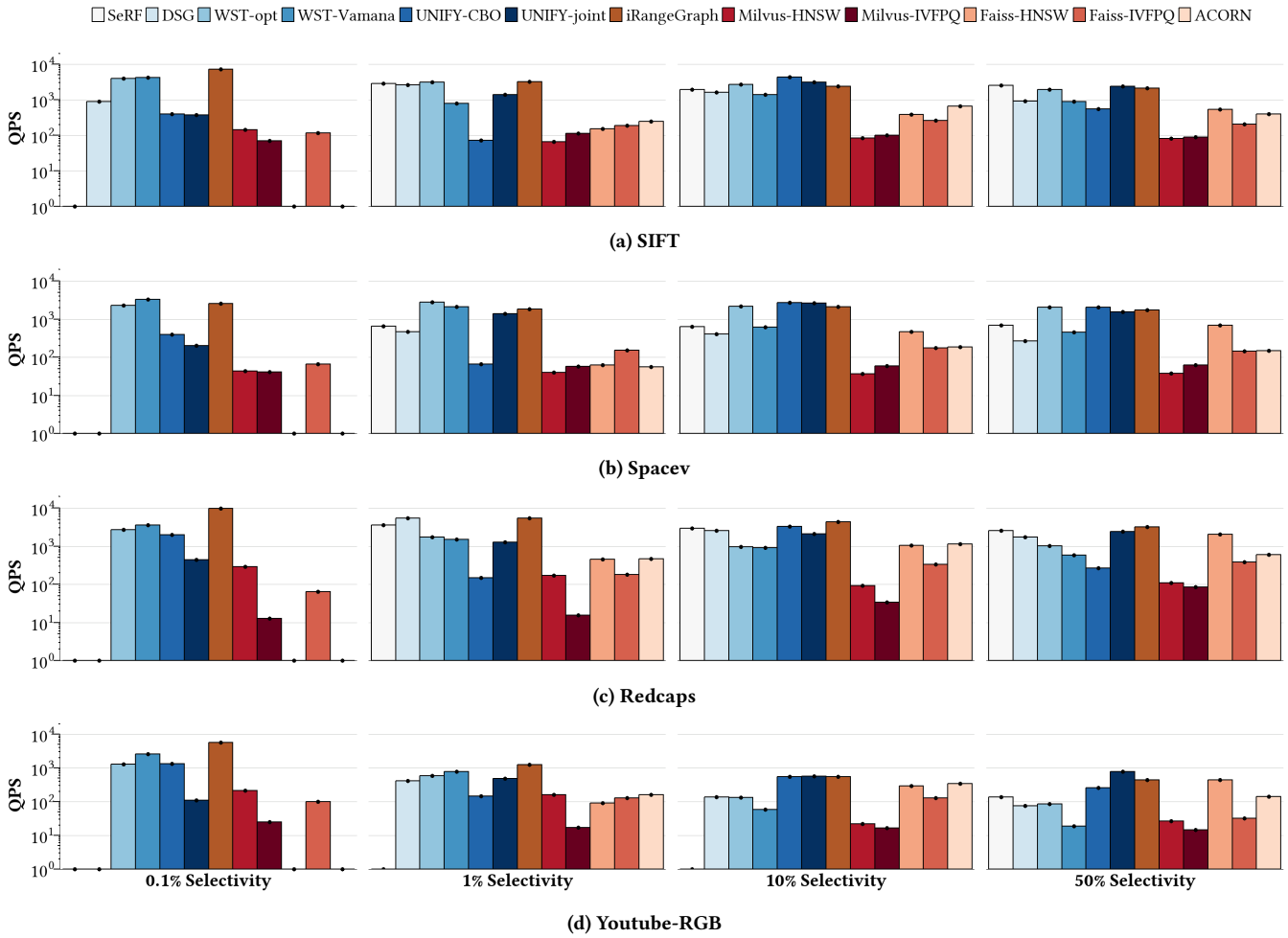


Figure 9: QPS for range Filtering ANN algorithms at 90% recall@10.

2. Scanning within a partitioned subset offers a simple yet effective strategy at low selectivity. Milvus-HNSW demonstrates strong performance at low selectivity levels. Interestingly, varying *ef_search* has no impact on its QPS or recall, suggesting that Milvus-HNSW performs direct scans within partitioned subsets rather than traversing the graph. In contrast, Faiss-HNSW and ACORN struggle at 0.1% selectivity, likely due to difficulties in identifying and reaching sparse candidates.

3. Multiple predicate filtering performs similarly to single predicate under uniform distribution. For uniformly distributed attributes, Faiss and ACORN handle multiple filtering predicates similarly, resulting in comparable performance between single and multiple predicates at the same overall selectivity. However, Milvus-HNSW exhibits different behavior (see point 2) due to changes in its internal search strategy when multiple predicates are applied.

5.5 Index Analysis

Table 5 reports index size (Size), search memory usage (Mem), and construction time (Time). Blue and red backgrounds indicate the best and worst performance under each filtering method. For Milvus, index size is not directly measurable due to compact storage; memory usage is estimated from total Docker consumption.

Compared to Faiss-HNSW, most algorithms exhibit higher index size, memory footprints and construction times. Overall, index size correlates positively with construction time (Pearson coefficient= 0.71).

Index size characteristics. Range filtering methods show significant higher index size than Faiss-HNSW, for their theoretical index sizes are $O(Mn\log(n))$, which is moderate. However, in practice, SeRF's memory usage is even lower, thanks both to its effective search performance with smaller M values and its tendency to skip many unnecessary ranges during index construction, resulting in a significantly reduced practical memory footprint.

In particular, WST-opt is the most memory-intensive, as it duplicates vector storage across overlapping BST subgraphs. These tradeoffs highlight the balance between filtering capability and storage efficiency, especially as dataset sizes increase.

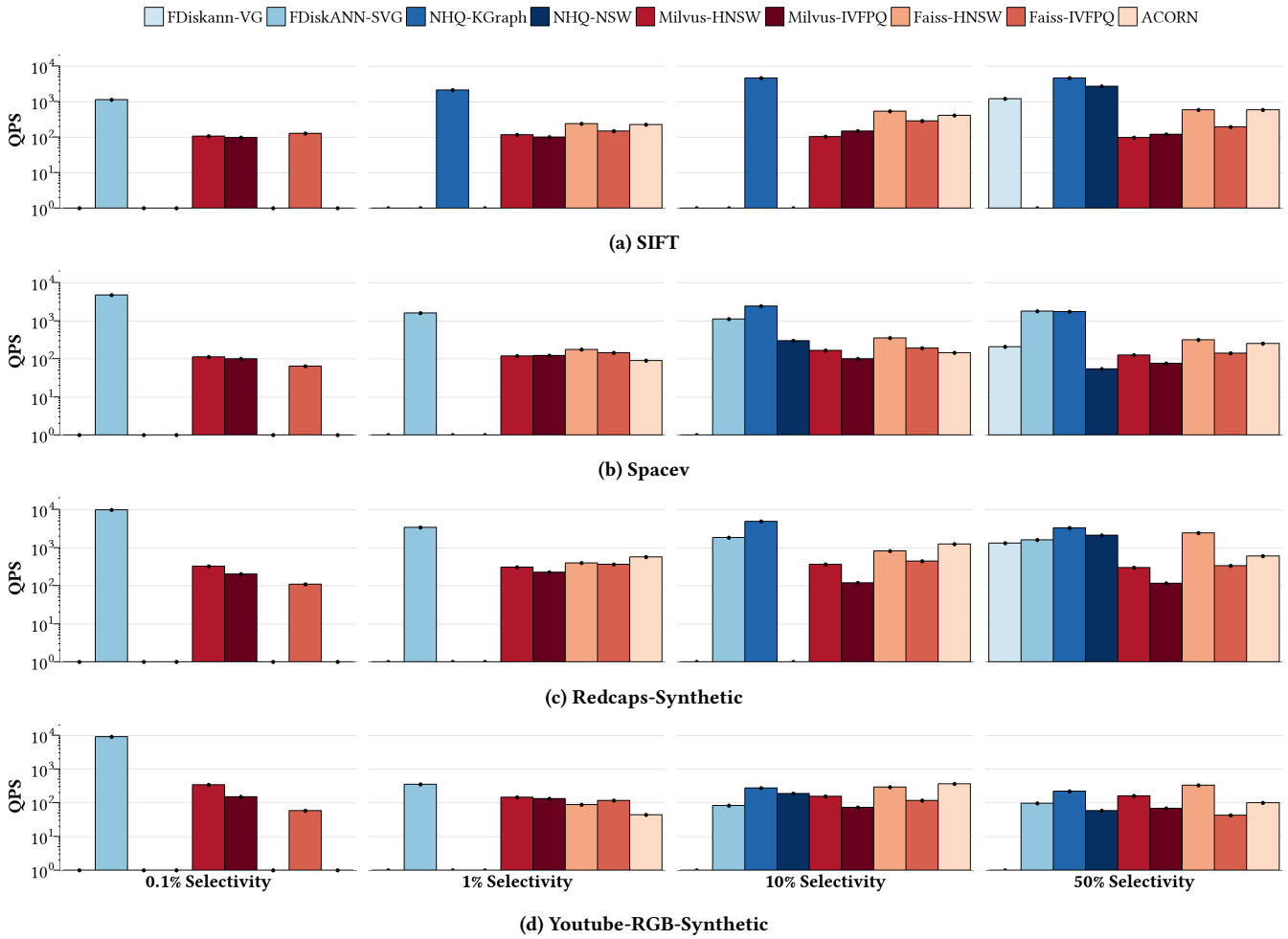


Figure 10: QPS for label Filtering ANN algorithms at 90% recall@10.

Memory characteristics. Filtering ANN methods typically require more memory than naive approaches.In containerized deployments like Milvus standalone, memory usage is strictly limited by Docker's preconfigured allocation regardless of dataset size. Quantization-based methods achieve better memory efficiency through compressed data representations. ACORN, Filtered-DiskANN and NHQ-NSW maintain similar edge counts to HNSW, resulting in comparable index size.

Some methods exhibit disproportionately high memory usage relative to their index sizes due to algorithm designs. For example, ACORN stores a boolean flag per vector for each query, and NHQ-KGraph duplicates the dataset unnecessarily.

Index construction time. Construction time is closely correlated with memory usage. An exception is iRangeGraph, which builds subgraphs sequentially (i.e., single-threaded), yielding high-quality indexes at the expense of parallel efficiency. Its reliance on RNG-based graph construction also results in slower indexing compared to methods using Vamana or KGraph. By contrast, SeRF maintains efficient construction times, primarily due to its operation with a small M parameter.

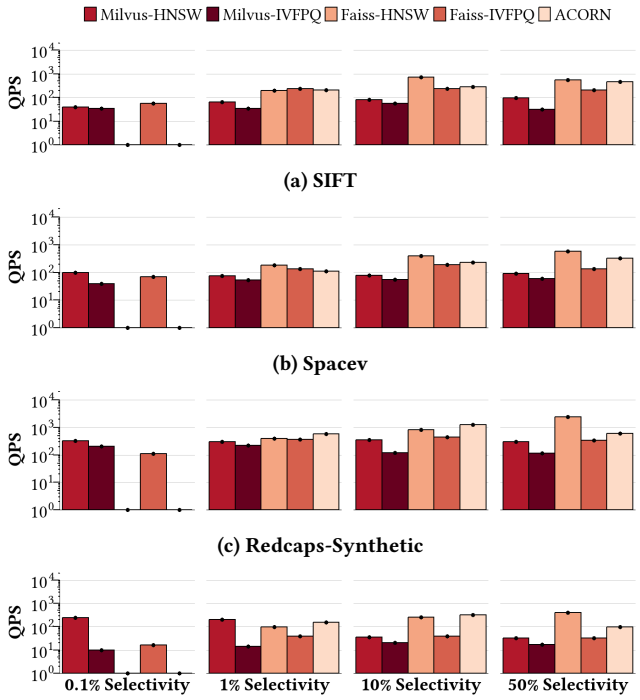
5.6 Pruning Strategy Evaluation

Several Filtering ANN algorithms, such as ACORN and Filtered-DiskANN, modify the standard RNG pruning strategy to improve connectivity under low-selectivity conditions. Similarly, NHQ-KGraph uses a KGraph-style pruning rather than NSW to achieve higher QPS, suggesting that RNG-style pruning may be suboptimal for low-selectivity scenarios. To evaluate this hypothesis, we modified both SeRF and ACORN to incorporate alternative pruning methods and compared their performance using comparisons as a detailed performance indicator. During construction step, ACORN_kg and SeRF_kg connect nearest neighbors directly, whereas ACORN_rng applies RNG-style pruning method.

Figure 12a shows the performance of different pruning methods across selectivity levels from 1% to 100% in ACORN (all methods failed at 0.1% selectivity). The results indicate that ACORN's original two-hop pruning performs best for its architecture with low

Table 5: Index size and index time.

Filtering Method	Algorithm	SIFT (4.80GB)			spacev (3.76GB)			Redcaps (1.91GB)			Youtube-RGB (3.81GB)		
		Size	Mem	Time	Size	Mem	Time	Size	Mem	Time	Size	Mem	Time
Arbi-	Faiss-HNSW	7.90	7.99	1733	6.86	6.93	2564	2.22	2.26	247	4.13	4.18	380
	Faiss-IVFPQ	0.68	2.90	1354	0.54	2.28	1370	0.25	1.27	1466	0.50	2.51	2165
	Milvus-HNSW	-	49.15	661	-	49.15	939	-	49.15	235	-	49.15	311
	Milvus-IVFPQ	-	49.15	507	-	49.15	621	-	49.15	323	-	49.15	548
	ACORN	10.45	109.33	3859	9.41	108.29	3321	2.48	12.39	810	4.38	14.31	757
Label-	FDiskANN-VG	6.34	7.24	505	5.30	6.34	465	2.07	2.19	35	3.97	4.12	31
	FDiskANN-SVG	6.35	7.11	370	5.30	6.27	325	2.07	2.16	210	3.97	4.00	100
	NHQ-KGraph	5.71	12.90	266	4.67	11.06	252	1.97	4.14	17	3.83	7.94	120
	NHQ-NSW	6.30	6.36	1658	5.26	5.31	1699	2.06	2.11	171	3.97	4.04	367
Range-	SeRF	6.96	10.44	364	5.88	8.35	374	2.09	3.93	61	3.89	7.76	45
	DSG	39.35	40.35	2905	36.88	37.87	3411	5.09	5.22	358	5.10	7.76	1045
	WST-opt	40.22	97.69	8048	34.68	92.66	8775	3.69	11.70	2391	4.20	17.41	1914
	WST-Vamana	23.62	59.09	4556	20.56	54.04	4839	2.91	9.07	1456	4.04	14.84	837
	iRangeGraph	24.27	44.72	69801	20.73	43.82	8455	3.33	5.30	6731	4.66	7.23	12155
	UNIFY	30.25	36.96	17721	29.21	34.88	24313	4.46	6.64	2308	6.36	10.48	3088



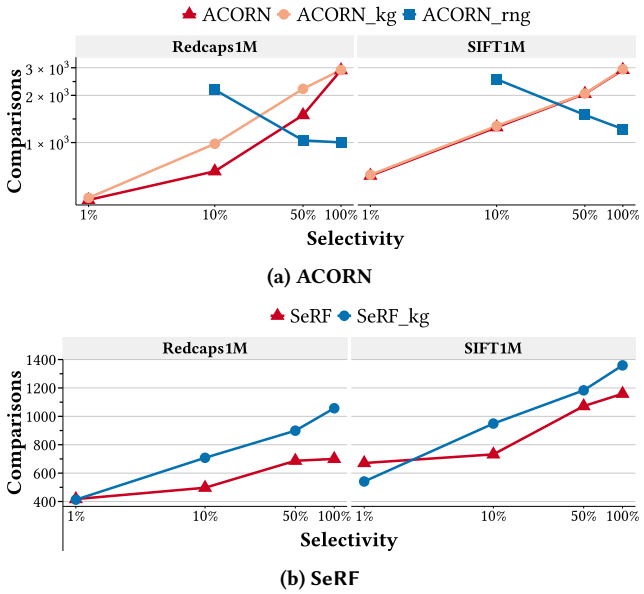
(a) SIFT

(b) Spacev

(c) Redcaps-Synthetic

(d) Youtube-RGB-Synthetic

Figure 11: QPS for arbitrary Filtering ANN algorithms at 90% recall@10.



(a) ACORN

(b) SeRF

Figure 12: Comparative performance of pruning methods at 90% recall@10 (Fewer Comparisons is better).

selectivity, while ACORN_kg achieves similar performance. Meanwhile, RNG performs better with above 50% selectivity, indicating different pruning strategies may suit different selectivity.

Figure 12b illustrates the impact of KGraph-style pruning on SeRF. While the original SeRF benefits from efficient edge construction through both RNG-style pruning and segmented edge filtering, SeRF_kg shows improved performance below 5% selectivity. However, it still fails to perform effectively at 0.1%. Overall, RNG-style pruning preserves graph quality at moderate to high selectivity but remains less effective in extremely selective cases.

5.7 Entry Point Selection Analysis

Entry point selection is a crucial component in graph-based ANN search. However, simple strategies often rival complex ones in Filtering ANN search. For example, SeRF and DSG select entry points

Table 6: Comparisons variation w.r.t. various entry point selection strategies on SIFT.

Algorithm	ep	Selectivity			
		1%	10%	50%	100%
UNIFY	**1**	**default**	**default**	**default**	**default**
UNIFY-B	3	-0.14%	0.10%	1.09%	2.21%
UNIFY-B	30	-0.32%	-0.83%	-0.91%	-1.42%
SeRF	**3**	**default**	**default**	**default**	**default**
SeRF	30	-3.34%	-2.52%	-2.28%	-1.34%
SeRF	300	-5.53%	-4.19%	-3.85%	-2.84%
DSG	**3**	**default**	**default**	**default**	**default**
DSG	30	-2.78%	-2.55%	-2.53%	-2.18%
DSG	300	-4.38%	-4.26%	-4.32%	-4.18%
UNIFY-B	**3**	**default**	**default**	**default**	**default**
UNIFY-B	30	-0.18%	-0.93%	-1.98%	-3.56%
UNIFY-B	300	-0.30%	-1.63%	-3.33%	-4.23%

from the bottom layer and still match the performance of hierarchical methods like UNIFY-joint. In this analysis, we investigate two factors: (1) the impact of hierarchical structures, and (2) the effect of varying the number of entry points in single-layer graph indexes.

UNIFY represents the optimal solution with a hierarchical structure. To evaluate the impact of hierarchy, we compare it with UNIFY-B, which retains only the bottom layer from UNIFY. Since SeRF and DSG are inherently single-layer indexes, we vary their entry point size (ep) to study the effect of the number of entry points. Table 6 presents the variation in Comparisons when adjusting entry point selection methods relative to each algorithm's default.

Effect of hierarchical structure. Comparing UNIFY with UNIFY-B reveals that the hierarchical structure has little impact on low-selectivity queries, but plays a more significant role as selectivity increases. We reckon that designing a method to reuse top-layer entry points in segmented edge methods, SeRF and DSG, might also lead to performance gains.

Effect of entry point size. Increasing the number of entry points for single layer indexes consistently reduces comparisons across all selectivity levels. UNIFY-B, SeRF, and DSG all perform better when using 30 or 300 entry points rather than just 3. Moreover, increasing the entry point set size improves search performance without recall loss. Notably, the performance of bottom-layer entry point selection with 30 or 300 points all surpass the default settings.

Notably, each method is compared with itself under different settings, so the number of comparisons directly reflects QPS. For example, in SeRF, using 300 entry points yields a 5.3% increase in QPS compared to the original method (which uses 3 entry points) at 50% selectivity, with a corresponding 3.85% decrease in Comparisons. Overall, incorporating more entry points properly is a worthwhile consideration for improving performance.

5.8 Edge Filtering Overhead Analysis

For most graph-based Filtering ANN search methods, comparisons per query show consistent performance, DSG show difference. Compared with SeRF, DSG performs fewer comparisons per query but

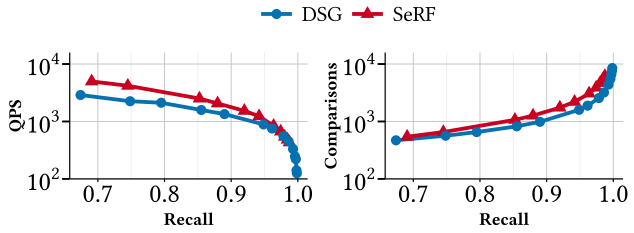


Figure 13: Recall/QPS and Recall/Comparisons for SeRF and DSG in SIFT at 10% selectivity.

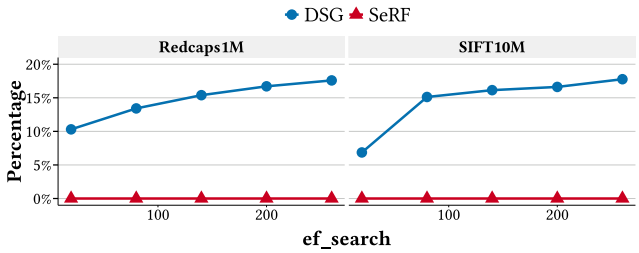


Figure 14: Edge filtering time (%) in SIFT at 10% selectivity.

achieves lower QPS, as shown in Figure 13. A potential reason is the additional overhead from filtering during neighbor selection. To validate this, we estimate the percentage of query time spent on neighbor selection.

Figure 14 demonstrates that DSG spends a significant amount of time finding valid edges, which becomes a key bottleneck. In contrast, SeRF spends almost no time on this step. As `ef_search` (the size of search candidate set) increases, the time required to filter valid edges also increases. This is mainly because, as the candidate set expands, it needs more time to find matched neighbors.

6 Lessons Learned

In this section, we summarize key insights (I) from our study, provide actionable recommendations, and outline promising open problems (O) for future study.

6.1 Insights

I1. Fine-grained segmented subgraphs are highly effective solutions to range Filtering ANN search. The most effective approach for range filtered ANN relies on subset indexing, typically implemented as segmented subgraphs (e.g., iRangeGraph, β-WST, UNIFY). These specialized structures efficiently address the challenges of filtered search by narrowing the search space.

I2. Segmented edge-based methods fail at low selectivity due to incomplete search during index construction. Methods like SeRF and DSG perform well only when their construction phase successfully connects relevant nodes. In low-selectivity settings, RNG-based techniques often miss such connections due to their monotonic search nature, indicating the need for alternative mechanisms that ensure connectivity.

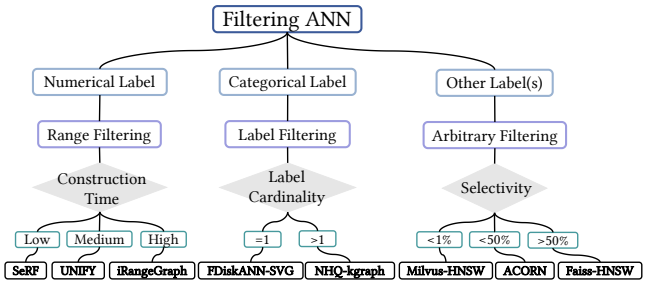


Figure 15: Guidebook for Filtering ANN algorithm usage.

I3. Partitioning is effective for low-selectivity queries. Milvus adopts partitioning to make low-selectivity queries viable. Similarly, more advanced techniques like stitching and segmented subgraphs further enhance performance in such settings.

I4. Label filtering algorithms remain underdeveloped. FDiskANN-SVG underperforms compared to NHQ-KGraph at high selectivity, mainly due to the weak quality of the Vamana graph. Conversely, NHQ-KGraph is unreliable at low selectivity. No method performs reliably across all settings, underscoring the need for better label filtering techniques.

I5. Hybrid distance filtering supports only high-selectivity queries. NHQ often fails at low selectivity because its connectivity is not guaranteed across all filtered subsets. Moreover, approaches involving multi-cardinality labels and strict matching are rarely used in practice due to their rigidity.

I6. Hierarchical structures offer limited benefits. Hierarchical indexing structures show effectiveness only under high selectivity. For example, UNIFY leverages a hierarchical design to sample entry points, but its advantage over the non-hierarchical variant becomes evident only when query selectivity exceeds 50%.

I7. Larger entry point sets improve performance. Increasing the number of entry points at the bottom layer consistently improves search performance across various algorithms and selectivity levels, often surpassing hierarchical structures.

6.2 Tool Selection

Figure 15 provides a practical guide for selecting Filtering ANN algorithms based on attribute type, index construction cost, cardinality, and selectivity.

For **numerical attributes**, UNIFY and iRangeGraph are strong options. iRangeGraph typically achieves the best query performance but requires longer index construction time. If construction efficiency or memory is a concern, SeRF offers a faster alternative, though it performs worse at low selectivity. Even though WST-opt achieves comparable performance to iRangeGraph, it is not recommended due to its significantly higher memory consumption. In contrast, while iRangeGraph has slower index construction, mainly because of its low parallelization, which is relatively easy to optimize.

For **label filtering**, FDiskANN-SVG is recommended when the query label cardinality is one. When cardinality exceeds one, NHQ-KGraph offers better performance.

In **arbitrary filtering** scenarios involving complex or mixed conditions, ACORN is the most flexible solution. However, for queries with high selectivity (>50%), traditional methods like Faiss-HNSW are often more efficient. For very low selectivity (<1%), Milvus-HNSW is preferred because of its stable and efficient scan strategy.

6.3 Open Problems

O1. Arbitrary Filtering ANN search remains an open challenge. ACORN is currently the only method designed for arbitrary filtering, yet it struggles with robustness across varying selectivity levels. The core difficulty lies in supporting arbitrary subset queries, which makes subgraph-based approaches (e.g., segmented or stitched methods) difficult to apply effectively. A promising direction is to explore IVF-based techniques, whose inherent partitioning structures may better support flexible and efficient subset search.

O2. Achieving optimal performance requires careful hyperparameter tuning. Filtering ANN search performance is highly sensitive to hyperparameters in both index construction and query execution. Optimal settings vary based on factors such as (1) dataset size, (2) vector distribution, (3) query selectivity, (4) size of K to be retrieved, and (5) intrinsic dimensionality for quantized methods. This makes universal tuning impractical. Future work should explore adaptive tuning strategies or develop robust algorithms that perform well without extensive manual calibration.

O3. Integration with vector databases remains limited. Most Filtering ANN methods rely on specialized index architectures tailored for specific filtering tasks (e.g., range or label filtering), creating barriers to integration with general-purpose vector databases. There is a clear need for hybrid indexing designs that retain compatibility with traditional ANN structures while supporting efficient filtering, enabling seamless adoption in real-world database systems.

O4. Dynamic Filtering ANN index is still underexplored. Among existing methods, only DSG supports dynamic index updates. Other range filtering approaches—such as SeRF, β-WST, iRangeGraph, and UNIFY—depend on attribute-sorted vector orders and do not support incremental insertions or deletions. Overcoming this limitation remains a fundamental challenge for making Filtering ANN algorithms suitable for dynamic environments.

O5. The attribute distribution analysis lacks. All generated attributes are assumed to follow a uniform distribution, a setting adopted by most existing methods. However, this assumption may not reflect real-world scenarios. Some works such as ACORN [50] and AIRSHIP [66] acknowledge this issue, but a comprehensive analysis of the interaction between vectors and attributes, as well as the performance of different methods under varying vector-attribute distributions, is still lacking.

7 Related Work

In addition to the algorithms and systems discussed in this paper, several other notable methods have been proposed. However, they are excluded from our experiments due to lower efficiency reported in prior studies or lack of open-source implementation.

Vector database systems. *AnalyticDB-V (ADBV)* [61] is a vector database system that supports vector search under arbitrary restrictions. ADBV employs a *Voronoi graph (VoG)* as an IVF structure to

reduce scan space and combines it with product quantization (PQ) for memory efficiency. A built-in cost estimator selects the most efficient strategy from among brute-force scan, PQ pre-filtering, VoGPQ pre-filtering, and VoGPQ post-filtering.

VBASE [65] is a vector database developed by Microsoft, extending PostgreSQL [1] with SQL-based vector search. It introduces a termination signal mechanism called *Relaxed Monotonicity*, which halts the search once retrieved candidates begin to diverge from the query vector. For filtering queries, VBASE softens the constraints to expand the search space, ensuring sufficient matched results.

Several other vector database systems are not included in our study, such as Vearch [5, 36], PASE [64], Weaviate [3], Pinecone [7], and Qdrant [8]. Among all vector database systems, Milvus is the most competitive method for its high efficiency, versatility [47] and widespread LLM applications [20]. Hence, we pick Milvus as the representative vector database system solution.

Filtering ANN algorithms. *RII* [42] introduced the concept of subset search, a precursor to filtered search. Built on IVFPQ, RII uses two scanning strategies based on a threshold: if the subset size is small, it scans the subset directly to get results; otherwise, it employs id-IVF, an index structure that maps IDs to clusters and skips clusters that do not contain target IDs. RII is omitted due to its naive filtering strategy, and its performance is almost identical to Faiss-IVFPQ according to our preliminary test.

AIRSHIP [66] is a label Filtering ANN method built on HNSW. It assumes that attribute distributions are biased and clustered, and samples vectors with diverse labels during index construction to ensure entry point coverage for all label types. During query execution, it performs both label-constrained and unconstrained searches using two separate candidate lists, making AIRSHIP effective under skewed attribute distributions. AIRSHIP was excluded from our experiments due to the unavailability of its open-source code.

8 Conclusion

In this paper, we present a comprehensive taxonomy and empirical study of filtering approximate nearest neighbor (ANN) algorithms, analyzing both algorithmic principles and implementation details. We categorize Filtering ANN methods based on the type of attributes, i.e., numerical (range filtering) and categorical (label filtering), and highlight the growing trend toward arbitrary filtering techniques for supporting arbitrary constraints. From a design standpoint, we classify filtering strategies into pre-filtering, post-filtering, and joint-filtering, each representing distinct trade-offs. For range filtering, we further examine implementation-level design choices that influence performance.

Our extensive experiments across diverse query scenarios show that segmented subgraph indexes consistently achieve superior performance in range filtering tasks. In contrast, label filtering remains unstable due to the lack of high-quality graph indexes. We identify two key factors limiting edge-based indexes: (1) pruning strategies and (2) entry point selection. Our study offers practical guidance for selecting suitable Filtering ANN methods and outlines open challenges that point toward promising directions for future research.

References

[1] 1996. PostgreSQL. https://www.postgresql.org/
[2] 2016. Yahoo. Nearest neighbor search with neighborhood graph and tree for high-dimensional data. https://github.com/yahoojapan/NGT
[3] 2019. Weaviate:Vector database for contextual queries. https://github.com/semi-technologies/weaviate
[4] 2020. Sptag: A library for fast approximate nearest neighbor search. https://github.com/microsoft/SPTAG
[5] 2020. Vearch: A Distributed System for Embedding-based. https://github.com/vearch/vearch
[6] 2021. *pgvector*. https://github.com/pgvector/pgvector
[7] 2021. Pinecone. https://www.pinecone.io/
[8] 2021. qdrant. http://qdrant.tech/
[9] 2025. The technical report is available in our supplementary material and the anonymous repository. https://anonymous.4open.science/r/FANNBench-41C0
[10] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *arXiv preprint arXiv:1804.06829* (2018).
[11] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.
[12] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. 2013. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company.
[13] BigANN Benchmark. 2021. Billion-Scale Approximate Nearest Neighbor Search Challenge: NeurIPS'21 competition track.
[14] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.
[15] Steve Dai, Rangha Venkatesan, Mark Ren, Brian Zimmer, William Dally, and Brucek Khailany. 2021. Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *Proceedings of Machine Learning and Systems* 3 (2021), 873–884.
[16] 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.
[17] Karan Desai, Gaurav Kaul, Zubin Aysola, and Justin Johnson. 2021. Redcaps: Web-curated image-text data created by the people, for the people. *arXiv preprint arXiv:2111.11431* (2021).
[18] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*. 577–586.
[19] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281* (2024).
[20] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).
[21] Joshua Engels, Benjamin Landrum, Shangdi Yu, Laxman Dhulipala, and Julian Shun. 2024. Approximate Nearest Neighbor Search with Window Filters. *arXiv preprint arXiv:2402.00943* (2024).
[22] Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228* (2016).
[23] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).
[24] 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.
[25] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.
[26] 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.
[27] Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. 2020. iDEC: indexable distance estimating codes for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 13, 9 (2020).
[28] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *arXiv preprint arXiv:2405.14831* (2024).
[29] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *arXiv preprint arXiv:2402.07630* (2024).

[30] Elias Jääsaari, Ville Hyvönen, and Teemu Roos. 2024. LoRANN: Low-Rank Matrix Factorization for Approximate Nearest Neighbor Search. *Advances in Neural Information Processing Systems* 37 (2024), 102121–102153.

[31] 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).

[32] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[33] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 861–864.

[34] Atsutake Kosuge and Takashi Oshima. 2019. An object-pose estimation acceleration technique for picking robot applications by using graph-reusing k-nn search. In *2019 First International Conference on Graph Computing (GC)*. IEEE, 68–74.

[35] Joseph B Kruskal. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society* 7, 1 (1956), 48–50.

[36] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyuan Ni, Ning Wang, and Yuan Chen. 2018. The design and implementation of a real time visual search system on JD E-commerce platform. In *Proceedings of the 19th International Middleware Conference Industry*. 9–16.

[37] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.

[38] Anqi Liang, Pengcheng Zhang, Bin Yao, Zhongpu Chen, Yitong Song, and Guangxu Cheng. 2024. UNIFY: Unified Index for Range Filtered Approximate Nearest Neighbors Search. *arXiv preprint arXiv:2412.02448* (2024).

[39] J MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press*.

[40] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[41] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[42] Yusuke Matsui, Ryota Hinami, and Shin'ichi Satoh. 2018. Reconfigurable Inverted Index. In *Proceedings of the 26th ACM international conference on Multimedia*. 1715–1723.

[43] Yitong Meng, Xinyan Dai, Xiao Yan, James Cheng, Weiwen Liu, Jun Guo, Benben Liao, and Guangyong Chen. 2020. Pmd: An optimal transportation-based user distance for recommender systems. In *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part II 42*. Springer, 272–280.

[44] 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.

[45] Lushuai Niu, Zhi Xu, Longyang Zhao, Daojing He, Jianqiu Ji, Xiaoli Yuan, and Mian Xue. 2023. Residual vector product quantization for approximate nearest neighbor search. *Expert Systems with Applications* 232 (2023), 120832.

[46] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1933–1942.

[47] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *The VLDB Journal* 33, 5 (2024), 1591–1615.

[48] Zhibin Pan, Liangzhuang Wang, Yang Wang, and Yuchen Liu. 2020. Product quantization with dual codebooks for approximate nearest neighbor search. *Neurocomputing* 401 (2020), 59–68.

[49] Rodrigo Paredes and Edgar Chávez. 2005. Using the k-nearest neighbor graph for proximity searching in metric spaces. In *String Processing and Information Retrieval: 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, November 2-4, 2005. Proceedings 12*. Springer, 127–138.

[50] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[51] Arkadiusz Paterek. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, Vol. 2007. 5–8.

[52] Zhencan Peng, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. [n. d.]. Dynamic Range-Filtering Approximate Nearest Neighbor Search. ([n. d.]).

[53] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.

[54] Patrick Schäfer, Jakob Brand, Ulf Leser, Botao Peng, and Themis Palpanas. 2024. Fast and Exact Similarity Search in less than a Blink of an Eye. *arXiv preprint arXiv:2411.17483* (2024).

[55] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.

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

[57] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[58] 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.

[59] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2024. An efficient and robust framework for approximate nearest neighbor search with attribute constraint. *Advances in Neural Information Processing Systems* 36 (2024).

[60] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631* (2021).

[61] 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.

[62] Yuexuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S Jensen. 2024. iRangeGraph: Improvising Range-dedicated Graphs for Range-filtering Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–26.

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

[64] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 2241–2253.

[65] 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.

[66] Weijie Zhao, Shulong Tan, and Ping Li. 2022. Constrained approximate similarity search on proximity graph. *arXiv preprint arXiv:2210.14958* (2022).

[67] 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.

Received 13 March 2025; revised 1 July 2025; accepted 24 August 2025