# An Experimental Evaluation of Hybrid Query Algorithms

Ben Trovato
Institute for Clarity in Documentation
Dublin, Ireland
trovato@corporation.com

Lars Thørväld
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Valerie Béranger
Inria Paris-Rocquencourt
Rocquencourt, France
vb@rocquencourt.com

Jörg von Ärbach
University of Tübingen
Tübingen, Germany
jaerbach@uni-tuebingen.edu
myprivate@email.com
second@affiliation.mail

Wang Xiu Ying
Zhe Zuo
East China Normal University
Shanghai, China
firstname.lastname@ecnu.edu.cn

Donald Fauntleroy Duck
Scientific Writing Academy
Duckburg, Calisota
Donald's Second Affiliation
City, country
donald@swa.edu

## ABSTRACT

Approximate Nearest Neighbor (ANN) search is a foundational technique in fields such as recommendation systems and information retrieval. However, in practical applications, users not only expect search results to be similar to the query in vector space but also to satisfy specific constraints, such as price ranges or categories. To meet these demands, researchers have proposed various hybrid query algorithms, including attribute filtering and range filtering. However, especially for attribute filtering, existing datasets often lack attribute values, and different researchers use varied testing configurations with simplified attribute value settings in their evaluations. Therefore, we select multiple real-world datasets and generate rich attribute values for various query scenarios, establishing a unified and comprehensive evaluation framework to compare these algorithms. Our study establishes a systematic taxonomy of hybrid query algorithms, exhaustively evaluates their time-space tradeoffs, performance, and robustness, and ultimately provides practical recommendations for different application scenarios.

**PVLDB Reference Format:**
Ben Trovato, Lars Thørväld, Valerie Béranger, Jörg von Ärbach, Wang Xiu Ying, Zhe Zuo, and Donald Fauntleroy Duck. An Experimental Evaluation of Hybrid Query Algorithms. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/zhujx001/Hybrid-ANNS.

## 1 INTRODUCTION

Nearest Neighbor (NN) search [15] aims to find the nearest vector in a given space and serves as a fundamental algorithm in vector retrieval, widely used in recommendation systems [22, 32, 33, 38, 39, 48] and image retrieval [49]. As data scales grow exponentially and vector dimensions increase, traditional NN search methods face the problem of "curse of dimensionality" [54]. This causes a sharp rise in computational cost. As a result, it becomes hard to meet the needs of real-time search. To solve this, researchers turn to more efficient methods—Approximate Nearest Neighbor (ANN) search [7, 11, 19, 35]. ANN search algotrithms build efficient index structures. It allows for a slight decrease in accuracy, but greatly enhances search efficiency. However, as application scenarios become more complex, simple ANN search is no longer sufficient to meet all practical needs. New requirements have emerged for ANN search systems. For example, as Figure 1, a user on an e-commerce platform may search for a dress by retrieving similar items based on an image. Additionally, the user can apply structured attribute filters such as price, color, or brand. This scenario necessitates a retrieval system that satisfies both vector similarity and attribute constraints [47]. When the constraint is a Boolean condition—e.g., the user specifically requires a green dress—the problem is referred to as Attribute Filtering Approximate Nearest Neighbor (AF-ANN) search [20, 51] problem. If the constraint is a range condition, such as filtering dresses priced between 100 and 200, it becomes the Range Filtering Approximate Nearest Neighbor (RF-ANN) search [56, 59] problem. To address such cases, hybrid query [29, 55] has been proposed. It jointly optimizes vector retrieval and conditional filtering. This approach significantly improves efficiency and flexibility in complex query scenarios.

### 1.1 Motivation

In recent years, hybrid query algorithms have advanced rapidly, with numerous algorithms proposed for attribute filtering and range filtering. In practice, the performance of hybrid query algorithms is also influenced by the characteristics of structured data. For attribute filtering algorithms, factors such as the number of base attributes, the number of query attributes, attribute distributions [12], and attribute selectivity [40] all affect actual performance. For range filtering algorithms, query range size and different datasets will affect the performance of the algorithm. Although the filter track of the BigANN 2023 competition [44] evaluated the performance of some algorithms, its assessment has several limitations: 1) The number of evaluated algorithms is limited, failing

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

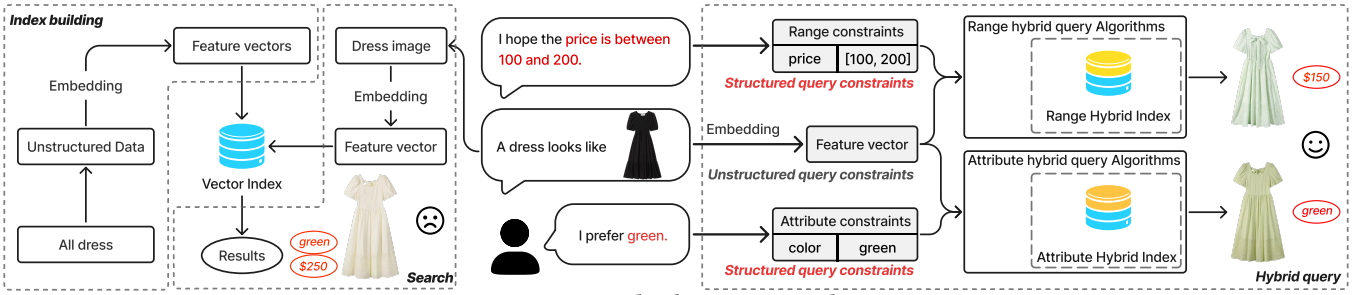



**Figure 1: Hybrid query example**

to cover mainstream methods. Mainstream attribute filtering algorithms such as NHQ and Filtered-DiskANN were not evaluated. 2) The evaluation focuses only on single-attribute and dual-attribute queries, without considering more complex multi-attribute scenarios. However, in actual applications, an object often involves multiple attributes, and the same is true for queries. For example, in e-commerce scenarios, users often specify multiple attributes (such as color, brand, price range, etc.) for search at the same time. 3) It did not include algorithm performance under different attribute distributions. Different distribution of attributes may have a direct impact on the filtering strategy and index construction of the hybrid query algorithm.

In addition, apart from the BigANN competition, no dedicated evaluation of hybrid query algorithms exists. To fill this gap, we conduct a comprehensive experimental evaluation of hybrid query algorithms, analyzing index construction overhead and query efficiency across diverse scenarios. In addition, we assess the robustness of each algorithm.

## 1.2 Our Contributions

This study investigates ANN search in hybrid query scenarios and presents a comprehensive review and evaluation of existing algorithms and systems. The main contributions include the following.

(1) **Systematic Classification and Overview.** Our study classifies over ten representative attribute-filtering algorithms from multiple perspectives, including index structure, filtering strategy, boolean logic support, and index construction method. It also incorporates range-filtering algorithms, widely-used vector libraries (e.g., Faiss), and vector databases (e.g., Milvus, PASE, VBASE), offering a unified reference framework for future research.

(2) **Evaluation of Attribute Filtering Algorithms.** Our evaluation covers several representative attribute-filtering algorithms on six real-world datasets. The experiments take into account factors such as the number of attributes, attribute distributions, and selectivity, in order to understand their impact on performance. The analysis includes index construction time, index size, peak memory usage, query speed (QPS), and accuracy. It further examines the robustness and adaptability of each method under complex conditions.

(3) **Evaluation of Range Filtering Algorithms.** We benchmarks five major range-filtering ANN algorithms on three large-scale datasets using different query ranges. The evaluation compares their efficiency in index construction, storage overhead, and query performance. It also analyzes how index structure influences algorithm performance, providing actionable insights for algorithm design.

(4) **Recommendations and Challenges.** Based on experimental results, the study offers algorithm recommendations for common use cases It identify key challenges in hybrid query. These include limited boolean logic support, lack of multi-attribute range filtering, high index cost of graph-based methods, and sensitivity to data distribution. Few current methods support both attribute and range filtering together, leaving an important gap for future work.

# 2 PRELIMINARIES

## 2.1 Problem Definition

We first define the Nearest Neighbor (NN) search problem.

*Definition 2.1 (NN Search).* Let $D = \{v_1, \ldots, v_n\}$ be a dataset of $n$ $d$-dimensional vectors. Given a query $Q = (q_v, k)$, where $q_v$ is the query vector and $k$ is a positive integer, the NN search aims to return a set $R \subseteq D$ with $|R| = k$, such that for any $x \in R$ and $y \in D \setminus R$, $dist(q_v, x) \leq dist(q_v, y)$. Here, $dist(\cdot, \cdot)$ denotes the distance metric, and we adopt Euclidean distance in this paper.

However, to address the curse of dimensionality faced by NN search [24], existing researches focus on approximate solutions, known as ANN search. We typically use Recall@$k = \frac{|R \cap \hat{R}|}{k}$ to evaluate the accuracy of ANN search algorithms, where $R$ denotes the true top-$k$ nearest neighbors of the query, $\hat{R}$ denotes the approximate top-$k$ nearest neighbors returned by the ANN search algorithm.

As the complexity of real-world application requirements increases, NN search has evolved into hybrid NN search with attribute constraints. Depending on the nature of the attribute constraints, hybrid NN search can be divided into two categories: 1) Attribute Filtering Nearest Neighbor (*AF-NN*) search. 2) Range Filtering Nearest Neighbor (*RF-NN*) search. We provide their formal definitions below.

*Definition 2.2 (AF-NN Search).* Let $D = \{(v_1, s_1), \ldots, (v_n, s_n)\}$ be a dataset of $n$ $d$-dimensional vectors, each associated with an attribute set $s_i$. Given a query $Q = (q_v, q_s, k)$, where $q_s$ is the query attribute set, the AF-NN search aims to return a set $R \subseteq D_s$ with $|R| = k$, such that for any $x \in R$ and $y \in D_s \setminus R$, $dist(q_v, x) \leq dist(q_v, y)$, where $D_s = \{v_i \mid (v_i, s_i) \in D \wedge q_s \subseteq s_i\}$.

We further classify AF-NN search according to the number of the attribute set $s_i$: 1) Single-Attribute Filtering Nearest Neighbor

(*SAF-NN*) search, where $\forall i, |s_i| = 1$. 2) Multi-Attribute Filtering Nearest Neighbor (*MAF-NN*) search, where $\exists i, |s_i| > 1$.

*Definition 2.3 (RF-NN Search).* Let $D = \{(v_1, s_1), \ldots, (v_n, s_n)\}$ be a dataset of $n$ $d$-dimensional vectors, each associated with an attribute value $a_i$. Given a query $Q = (q_v, [a_{\min}, a_{\max}], k)$, where $a_{\min}$ and $a_{\max}$ denote the lower and upper bounds of the query range, respectively, the RF-NN search aims to return a set $R \subseteq D_a$ with $|R| = k$, such that for any $x \in R$ and $y \in D_a \setminus R$, $dist(q_v, x) \leq dist(q_v, y)$, where $D_a = \{v_i \mid (v_i, a_i) \in D \wedge a_{\min} \leq a_i \leq a_{\max}\}$.

Similar to the conventional ANN search, most existing studies focus on approximate solutions for hybrid NN search, referred to as hybrid ANN search, which includes both *AF-ANN* search and *RF-ANN* search.

## 2.2 Index Structures in ANN Search

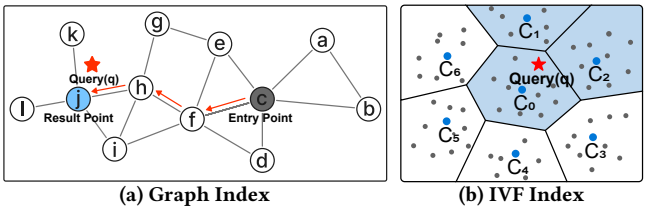
Current hybrid query methods mainly adopt graph-based [16, 18, 23, 25, 34] or Inverted File Index (IVF)-based [28] index structures. We briefly introduce the fundamental principles of these two types in the following.

**Graph.** Graph-based ANN search algorithms accelerate queries by building graph indexes on datasets. As illustrated in Figure 2a, each data point in the dataset corresponds to a point in the graph, and neighboring vertices (e.g., $a$ and $b$) are connected via edges based on their distance $dist(a, b)$. In the graph index, each point maintains connections only to its nearest neighbors to preserve query efficiency. For instance, in Figure 2a, the gray point $c$ is connected to five neighbors $(a, b, d, e, f)$ and can traverse to them via corresponding edges.

Given a query point $q$, the graph-based ANN search typically follows a greedy search strategy to find the $k$ nearest neighbors. For illustration, consider the case where $k = 1$ (i.e., finding the single nearest neighbor of $q$): The algorithm initiates the search from an entry point (the gray point $c$). If there exists a neighbor $n$ (the point $f$) of the current point such that $dist(n, q) < dist(c, q)$, the algorithm updates the current point to $n$. This process is iteratively repeated—always moving to the neighbor closest to $q$—until no neighbor is found that is closer than the current point. The final point reached (the blue point $j$) is returned as the approximate nearest neighbor of $q$.

**IVF.** IVF-based ANN search algorithms improve computational efficiency by partitioning the dataset into clusters and restricting the search to clusters nearest to the query point. Specifically, as shown in Figure 2b, IVF first selects a subset of data points and applies a clustering algorithm (e.g., K-Means) to obtain a set of centroids $(C_0, C_1, ..., C_6)$, each representing a distinct cluster. Every data point (gray dots) is then assigned to the cluster corresponding to its nearest centroid based on the distance metric, thereby forming the index structure.

During the query process, given a query point $q$, the algorithm computes distances between $q$ and all centroids, selects a small number of the closest centroids $(C_0, C_1, C_2)$, and then performs a search only within the associated clusters to identify the approximate nearest neighbors.




**(a) Graph Index** **(b) IVF Index**

**Figure 2: Graph index and IVF index**

# 3 OVERVIEW OF HYBRID QUERY ALGORITHMS

We provide an overview and analysis of over ten hybrid query methods, all drawn from recent research efforts, and collectively referred to as algorithms in the following text. Specifically, Section 3.1 describes methods based on attribute filtering, Section 3.2 discusses those based on range filtering, and Section 3.3 introduces vector libraries and databases that support both attribute and range filters.

## 3.1 Attribute Filtering Algorithms

We summarize six attribute filtering algorithms and provide an intuitive comparison in Table 1.

**NHQ [51].** Traditional attribute filtering algorithms usually perform attribute constraints and ANN search separately. In contrast, NHQ is the first to implement simultaneous filtering, integrating both aspects into a unified framework. NHQ constructs an index based on a nearest neighbor graph and introduces a fusion distance that jointly captures vector similarity and attribute similarity. Leveraging this fusion distance, NHQ unifies vector similarity and attribute matching into a single comprehensive similarity measure and builds the graph accordingly. During the query process, NHQ efficiently prunes irrelevant edges via this composite index, enabling fast retrieval of results that satisfy both vector similarity and attribute constraints.

**Filtered-DiskANN [20].** The effectiveness of NHQ may degrade when the number of attributes changes, as it affects the fusion distance computation. Filtered-DiskANN addresses this limitation.

Filtered-DiskANN also supports simultaneous filtering. Built upon the Vamana [26] graph-based ANN index, it incorporates attribute information directly into the graph during index construction. This integration ensures that the index reflects both vector proximity and attribute constraints. Moreover, Filtered-DiskANN supports SSD-based storage, improving scalability.

Filtered-DiskANN proposes two index variants: 1) **FilteredVamana**, which incrementally builds the graph index by inserting data points and dynamically adding edges, allowing adaptive expansion. 2) **StitchedVamana**, which adopts a batch construction strategy—constructing separate Vamana subgraphs for each attribute, followed by merging and edge pruning.

A key limitation of Filtered-DiskANN is that while it supports both single-attribute and multi-attribute filtering, it only supports the Boolean OR logic for multi-attribute filtering, lacking support for Boolean AND logic.

**Table 1: Comparison of AF-ANN search algorithms**

| Algorithm | Base | Filter Type | AND | OR | Flexible Attributes | Complex Boolean | Dynamic Insert | Multi Thread |
|---|---|---|---|---|---|---|---|---|
| NHQ | Graph | C | Y | N | N | N | N | N |
| FilteredVamana | Graph | C | N | Y | Y | N | Y | Y |
| StitchedVamana | Graph | C | N | Y | Y | N | N | Y |
| CAPS | IVF | C | Y | N | N | N | Y | Y |
| ACORN | Graph | C | Y | Y | Y | Y | Y | Y |
| UNG | Graph | B | Y | Y | Y | N | Y | Y |
| Puck | IVF | B | Y | Y | Y | N | Y | Y |

[A] Post-filtering, [B] Pre-filtering, [C] Simultaneous filtering, [Y] Support, [N] Unsupport. *Base* indicates that the hybrid query algorithm is an improvement based on a specific ANN search algorithm. *AND* indicates whether the algorithm supports attribute filtering with AND operations. *OR* indicates whether the algorithm supports attribute filtering with OR operations. *Flexible Attributes* indicates whether the number of query attributes can differ from the number of attributes used during index construction. *Complex Boolean* refers to whether the algorithm supports advanced Boolean logic beyond just AND and OR, such as NOT or combinations like (A AND B) OR (C AND NOT D). *Dynamic* indicates whether the algorithm supports the operation of dynamically inserting data points.

**CAPS [21].** As graph-based methods, NHQ and Filtered-DiskANN often result in large index sizes. CAPS addresses this issue by being the first simultaneous filtering algorithm based on spatial partitioning, significantly reducing index size.

CAPS introduces a hierarchical sub-partitioning algorithm inspired by Huffman trees, termed the Attribute Frequency Tree (AFT), to overcome the coarse granularity of traditional IVF-based methods. CAPS adopts a two-level partitioning strategy: 1) The first level clusters vectors based on similarity using K-Means or learning-based methods such as BLISS. 2) Within each cluster, AFT partitions data further based on attribute frequencies, enabling finer-grained indexing and improving query efficiency.

**ACORN [40].** Above simultaneous filtering methods struggle with large-scale, unbounded, or unknown predicate sets. ACORN addresses this issue by introducing a predicate-agnostic indexing framework.

ACORN is built upon HNSW [36], a hierarchical graph-based ANN index. It supports high-cardinality and unrestricted predicate sets, overcoming the limitations of methods restricted to small-scale equality filters. It constructs a denser hierarchical graph by expanding node neighborhoods and prunes edges at lower levels to control index size. During querying, ACORN eliminates unnecessary distance computations by filtering out nodes that violate attribute constraints. This effectively maintains a nearest neighbor graph over valid nodes only.

ACORN includes two index variants: 1) **ACORN-$\gamma$**, which expands neighbor lists during construction, increasing memory usage for higher performance. 2) **ACORN-1**, which extends neighbor lists by considering second-hop neighbors during search, reducing index size with minor performance loss.

**UNG [12].** Graph-based methods (NHQ, Filtered-DiskANN, ACORN) do not guarantee result completeness, while CAPS lacks robustness when base and query vectors differ in attribute cardinality. UNG is proposed to overcome both limitations.

UNG is a unified framework that integrates diverse graph-based ANN indexes for hybrid query. It first groups the dataset by attribute sets, ensuring that vectors in each group share identical attributes. Then, it constructs a Label Navigating Graph (LNG) to encode inclusion relationships among attribute sets. Within each group, UNG builds graph-based ANN indexes (e.g., Vamana, HNSW) and connects them via cross-group edges to enable efficient cross-group search.

**Table 2: Comparison of RF-ANN search algorithms**

| Algorithm | Base | Dynamic Insert | Multi Thread |
|---|---|---|---|
| DSG | Graph | Y | N |
| iRange | Graph | N | Y |
| SeRF | Graph | Y | Y |
| UNIFY | Graph | Y | Y |
| WinFilter | Graph | N | Y |

[Y] Support, [N] Unsupport. *Base* indicates that the hybrid query algorithm is an improvement based on a specific ANN search algorithm. *Dynamic* indicates whether the algorithm supports the operation of dynamically inserting data points.

UNG supports Boolean filtering with two modes: 1) The query attribute set is a subset of the data attribute set. 2) The query attribute set exactly matches the data attribute set.

**Puck [9].** While simultaneous filtering performs well in most scenarios, it may underperform when only a small portion of data satisfies attribute constraints. In such cases, a pre-filtering strategy—applying attribute constraints prior to ANN search—can be more effective.

Developed by Baidu, Puck utilizes two-level quantization for indexing. It maintains an attribute set for each cluster to track vector attributes. During the query process, it employs pre-filtering to exclude clusters without required attributes, thereby reducing the search space.

## 3.2 Range Filtering Algorithms

We introduce five representative range filtering algorithms and present a comparative overview in Table 2.

**SeRF [59].** Conventional RF-ANN search approaches typically follow one of two paradigms: 1) conducting ANN search first followed by attribute-based filtering; 2) filtering the dataset based on attribute ranges before performing ANN search. However, both approaches suffer from suboptimal performance. Building a separate neighbor graph (e.g., HNSW) for each attribute range could ensure efficient querying but incurs an $O(n^2)$ cost in constructing and storing $n$ graphs. Since many edges are shared across these graphs, SeRF introduces a validity-range aware design where each edge records the interval in which it is valid, indicating in which subgraphs the edge remains valid. This compresses $n$ graphs into a single unified graph, maintaining search effectiveness while significantly reducing memory and index construction overhead.

**WinFilter [52].** Unlike SeRF, which focuses on graph compression, WinFilter proposes a structural partitioning framework

called the $\beta$-Window Search Tree ($\beta$-WST). After the dataset is sorted by attribute values, it is partitioned into multiple intervals and organized into a tree structure. Each node in the tree corresponds to an attribute range and maintains a local ANN index (e.g., Vamana). For a range-filtering query, WinFilter only searches within nodes overlapping the query range and merges partial results. With a tree height of $O(\log n)$, each query accesses at most $O(\log n)$ sub-indexes, resulting in significant speedups.

**iRange [56].** To address the query efficiency degradation caused by the graph compression in SeRF, iRange adopts a more flexible strategy. Rather than prebuilding indexes for all possible ranges, it dynamically assembles a query-specific subgraph at runtime. iRange partitions the dataset into intervals based on attribute values and independently constructs a local graph for each interval, storing only edge information. During the query process, the relevant local graphs overlapping with the query range are merged into a temporary search graph, and a pruning strategy is applied to enable efficient search.

**DSG [41].** Most RF-ANN search methods, such as iRange and WinFilter, are designed for static datasets. SeRF allows incremental insertion but requires attributes to be ordered. DSG introduces the first truly dynamic RF-ANN framework that supports inserting data with unordered attributes while maintaining efficient range filtering. DSG is built on two key data structures. 1) The rectangle tree partitions the query space into rectangular regions, each corresponding to a group of queries that share the same nearest neighbors. 2) The dynamic segment graph is a neighbor graph where each edge is annotated with the attribute range in which it is valid. With these structures, only a few regions need to be updated when inserting new data. During queries, the system considers only edges valid for the current attribute range, improving efficiency.

**UNIFY [31].** Unlike DSG, which relies on rectangle trees and range-aware edges, UNIFY adopts a segmentation-based approach and proposes a range query index structure. UNIFY introduces the Segmented Inclusive Graph (SIG), which partitions the dataset into segments by attribute and constructs an independent neighbor graph for each segment. These segment graphs are then integrated into a unified global graph. SIG adheres to a graph inclusiveness principle: any query subgraph can be composed from existing segment graphs, avoiding the need for range-specific index construction. Its hierarchical variant, HSIG, enhances SIG by incorporating the multilayer design of HNSW, skip lists, and edge bitmaps for efficient range localization and post-filter pruning. UNIFY supports three filtering strategies—pre-filtering, post-filtering, and simultaneous filtering—enabling robust and flexible performance across diverse application scenarios.

## 3.3 Vector Libraries and Databases

The vector libraries and databases discussed in this section are not explicitly designed for hybrid query, but they support both attribute filtering and range filtering. Additionally, all the functionalities shown in Table 1 are also supported.

**Faiss [17].** Faiss is a library designed for efficient similarity search and clustering of dense vectors. It supports various indexing structures, including IVF, Product Quantization (PQ) [28], and HNSW.

In hybrid query scenarios, Faiss supports vector search with an ID selector, which is a bitmap aligned with the dataset size. Users can generate this selector by first applying custom attribute filtering methods. During the query, Faiss excludes vectors based on the selector, enabling efficient integration of attribute filtering with ANN search.

In addition, we adopted the batch optimization strategy of HQI [37] for the IVF index in Faiss. This strategy groups queries with the same filter conditions, performs filtering once per group, and then uses efficient matrix operations to perform batch vector similarity calculations.

**PASE [57].** PASE is a vector indexing plugin for the PostgreSQL database [43], supporting two index types: IVF_Flat [27] and HNSW. By leveraging database capabilities, PASE enables both attribute filtering and range filtering. We focus on the post-filtering strategy based on HNSW.

When a query request is received, PASE first obtains a candidate set using the HNSW index. It then applies the filtering conditions from the WHERE clause to refine the results. However, since the candidate set has a fixed size, low-selectivity filters may result in fewer than $k$ results being returned.

**VBASE [58].** Similar to PASE, VBASE is a PostgreSQL-based vector indexing plugin supporting HNSW, SPTAG [13], and SPANN [14]. We focus on HNSW-based search.

VBASE adopts a post-filtering strategy. However, it employs an iterative filtering mechanism: during index traversal, each node is immediately checked against the WHERE clause, and non-matching nodes are discarded. This avoids the issue in PASE where the final result set may contain fewer than $k$ results.

**Milvus [50].** Unlike PASE and VBASE, which are database extensions, Milvus is an open-source vector database purpose-built for large-scale similarity search. It supports multiple index types, including HNSW, IVF_Flat, and IVF_PQ, and provides extensive optimization for real-world deployment.

Among its indexes, IVF_Flat is commonly used due to its balance between performance and simplicity. In hybrid queries, Milvus first pre-filters the dataset using user-specified conditions, then traverses only the corresponding clusters for ANN search—similar to the ID selector mechanism of Faiss.

# 4 EXPERIMENTS

## 4.1 Experimental Setup

*4.1.1 Datasets.* For attribute filtering tasks, we employ six real-world datasets widely adopted in existing literature: Msong [5], Audio [6], SIFT1M [4], GIST1M [4], GloVe [42], and Enron [45], covering a diverse range of domains. We independently generate attribute for each dataset.

For range filtering queries, we evaluate three widely used real-world datasets: Deep [10], YT-Audio [2], and WIT [1]. Following

**Table 3: Datassets**

| Dataset | Dimension | Base Data | Queries | LID | Type |
|---|---|---|---|---|---|
| Msong | 420 | 992,272 | 200 | 23 | Audio |
| Audio | 192 | 53,387 | 200 | 14 | Audio |
| SIFT1M | 128 | 1,000,000 | 10,000 | 19 | Image |
| GIST1M | 960 | 1,000,000 | 1,000 | 45 | Image |
| GloVe | 100 | 1,183,514 | 10,000 | 47 | Text |
| Enron | 1369 | 94,987 | 200 | 23 | Text |
| Deep | 96 | 1,000,000 | 10,000 | 22 | Image |
| YT-Audio | 128 | 1,000,000 | 10,000 | 15 | Audio |
| WIT | 2048 | 1,000,000 | 40,300 | 48 | Image |

prior work [41], we use the data point ID within each dataset as its attribute. This ensures that the data is initially sorted.

Table 3 summarizes the key characteristics of all datasets. In particular, we report the Local Intrinsic Dimensionality (LID) [30], a commonly used metric to quantify dataset hardness. Following the standard evaluation protocol [3], we randomly sample 10,000 data points from each dataset for LID computation. A higher LID indicates greater intrinsic complexity.

*4.1.2 Evaluation Metrics.* To comprehensively assess the overall performance of different algorithms, we adopt a multi-dimensional quantitative evaluation framework [53], which includes the following five core metrics:

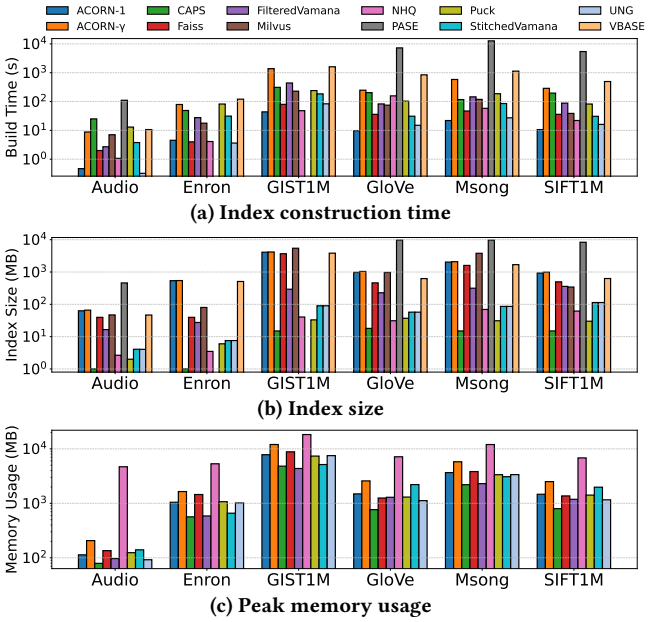
(1) **Recall@k**: The proportion of overlap between the returned approximate $k$ nearest neighbors and the ground-truth $k$ nearest neighbors.
(2) **QPS (Queries Per Second)**: The number of queries processed per second.
(3) **Index Construction Time**: The total time required to transform the raw dataset into a queryable index structure.
(4) **Index Size**: The storage size of the persisted index on disk.
(5) **Peak Memory Usage**: The maximum memory consumption observed during the index construction phase.

*4.1.3 Parameter Settings.* For all evaluated algorithms, we adopt the parameter settings recommended in the original paper and adjust the parameters related to search in order to obtain different recall and QPS.

*4.1.4 Platform.* We conduct all experiments on a server running Ubuntu 20.04, equipped with an AMD EPYC 7K62 processor (2.6GHz) and 256GB of RAM. Unless otherwise stated, we execute index construction with 32 threads to accelerate the process [8]. By default, queries run on a single thread; we also report results for 16-thread parallelism in specific scenarios.

We summarize the multi-threading capabilities of each algorithm in Table 1 and Table 2. For vector database systems, we adopt a multi-process evaluation scheme (rather than multi-threaded execution), following the methodology proposed in VectorDBBench [46], to better reflect their real-world performance characteristics.

## 4.2 Attribute Filtering



(a) Index construction time
(b) Index size
(c) Peak memory usage

**Figure 3: Time and space overhead of attribute filtering index construction**

*4.2.1 Time and Space overhead of index construction.* To investigate the performance of different attribute filtering algorithms during index construction, we evaluate three key metrics in the single-attribute scenario: index construction time, index size, and peak memory usage. Notably, PASE is evaluated exclusively on Audio, GloVe, Msong, and SIFT1M as it supports only data with dimensionality less than 512.

***Index construction time.*** As shown in Figure 3a, among the aforementioned four datasets, PASE has the longest index construction time due to its single-threaded indexing support. Additionally, VBASE also supports only single-threaded indexing but achieves better query performance than PASE.

For algorithms supporting 32-thread, index construction time differences are negligible on small-scale datasets (Audio and Enron) due to their limited complexity. On large-scale datasets (SIFT1M, GIST1M, GloVe, and Msong), ACORN-1 achieves the fastest index construction. This efficiency stems from its similarity to the original HNSW construction process and the use of a limited number of candidate neighbors. In contrast, ACORN-$\gamma$ incurs the highest index construction time, as it expands neighbor lists and evaluates a large candidate pool, leading to increased computational overhead.

***Index size.*** As illustrated in Figure 3b, across all datasets, the two IVF-based methods (CAPS and Puck) yield significantly smaller index sizes compared to graph-based approaches. CAPS consistently produces the most compact index across all datasets, leveraging its spatial partitioning mechanism.

***Peak memory usage.*** As shown in Figure 3c, on small-scale datasets, all methods except NHQ exhibit low memory usage. The substantially higher memory consumption of NHQ is attributed to its graph construction procedure, which requires loading the entire dataset into memory. On large-scale datasets, CPAS exhibits

slightly lower memory usage compared to other algorithms. Due to their special characteristics, databases were not included in the peak memory usage testing during index construction.

*4.2.2 Performance Evaluation.* We next evaluate the query performance of AF-ANN search algorithms, focusing on three main scenarios.

***Single-Attribute Building and Single-Attribute Query.*** In this scenario, we construct the index using a single attribute and issue queries on the same attribute.

*Single-Threaded Search.* As shown in Figure 4, under full-recall conditions (recall = 1), search often requires traversing larger candidate sets or deeper graph paths, leading to increased computational cost and a sharp QPS drop for most methods. Despite this, UNG maintains high QPS due to its LNG structure, which eliminates unnecessary online filtering of irrelevant attributes, thereby improving query efficiency. But, UNG slightly underperforms on high-LID datasets such as GloVe. In contrast, CAPS performs better on high-LID datasets but is more sensitive to dataset characteristics. CAPS leverages multi-level spatial partitioning and attribute filtering to efficiently explore sparse spaces, whereas UNG relies on structured attribute organization, which limits its adaptability to sparse high-dimensional distributions.

NHQ, StitchedVamana, and FilteredVamana show similar and stable performance across all datasets. This indicates that joint filtering and search strategies can achieve consistent effectiveness under high-recall requirements. As observed in Figure 4, StitchedVamana outperforms FilteredVamana in query efficiency. The difference arises from their pruning strategies: StitchedVamana applies pruning after merging graphs, allowing nodes to accumulate a richer candidate set; while FilteredVamana applies early pruning, potentially eliminating useful candidates prematurely.

Among all methods, vector database systems (PASE, VBASE, Milvus) perform the worst. These systems target general-purpose scenarios but incur communication latency from network I/O and experience high query processing overhead due to complex query parsing. Additionally, we observe that Faiss+HQI_Batch outperforms vanilla Faiss, indicating that batch querying via HQI can effectively enhance query performance.

*Multi-Threaded Search.* As shown in Figure 5, UNG continues to achieve the best performance. Puck performs well on large-scale datasets, but its performance degrades on small datasets. Puck is tailored for large-scale scenarios, and optimizations such as two-level inverted indexing and hierarchical quantization introduce unnecessary overhead on small datasets. CAPS maintains a relatively high throughput under multi-threading but exhibits limited adaptability to different datasets. The vector distribution of the dataset affects the construction of its partitioned index.

ACORN performs poorly overall, primarily due to the sensitivity of its hyperparameter $\gamma$ to attribute selectivity. Without careful tuning based on dataset characteristics, it is challenging to construct an effective index.

***Multi-Attribute Building and Single-Attribute Query.*** In this scenario, the index is constructed using multiple attributes but applies filtering condition on only one attribute during query processing.

In database systems (VBASE, PASE, Milvus), when the system builds both B+ tree and vector indexes, queries typically utilize only the B+ tree. Faiss offers an ID filtering mechanism, it operates solely during query execution and does not influence index building. CAPS requires that the number of attributes in a query match those used during index construction. ACORN-1 and ACORN-$\gamma$ are specifically designed for single-attribute indexing. Hence, they are excluded from this comparison.

*Effect of Multi-Attribute Building on Algorithm Performance (M vs. S).* As shown in Figure 6, Puck demonstrates nearly no performance degradation. In contrast, the performance of StitchedVamana, FilteredVamana, NHQ, and UNG drops significantly, with the following reasons: FilteredVamana increases the complexity introduced by retaining attribute during index construction; the complexity of StitchedVamana increases because data points may duplicate across multiple subgraphs; The fusion distance of NHQ exhibits sensitivity to the number of attributes, which degrades its effectiveness; UNG employs a distinct construction and search strategy, leading to a significant drop in query performance.

*Performance of Algorithms under Multi-Attribute Building (M vs. M).* As shown in 6, Puck achieves the best query performance, surpassing other algorithms by an order of magnitude. It filters out irrelevant results during retrieval rather than searching for more candidates. FilteredVamana performs the worst because its multi-attribute indexing forces neighbor selection to cover multiple attributes, hurting single-attribute query efficiency. This leads to excessive irrelevant node traversal, making it slower than other methods.

***Multi-Attribute Building and Multi-Attribute Query.*** Compared to single-attribute filtering, multi-attribute joint filtering better reflects real-world application scenarios. To evaluate algorithm performance under such scenario, we construct the index using three uniformly distributed attributes and apply filtering conditions on all three attributes simultaneously during the query process.

As shown in Figure 7, UNG achieves the best performance across all datasets. For IVF-based methods that are not specifically optimized for hybrid query scenarios (Faiss_IVF and Milvus), their QPS remains relatively stable as the recall varies. Milvus, being a general-purpose system, exhibits comparatively poor performance. In contrast, Faiss—especially when combined with HQI_Batch—achieves moderate performance. Its original IVF implementation naturally supports pre-filtering and does not suffer from the complexity introduced by an increased number of attributes. In fact, filtering can reduce the effective candidate set, improving efficiency.

The IVF-based algorithm CAPS, optimized for hybrid query scenarios, employs a multi-level partitioning strategy that introduces additional overhead. This overhead negatively impacts performance on small datasets. However, on large datasets, the overhead is amortized, and the benefits of attribute-aware partitioning become evident, making CAPS the second-best performer after UNG.

NHQ exhibits mediocre performance overall and performs particularly poorly on small datasets. It relies on fused distance calculations. The fusion distance computation is highly sensitive to the

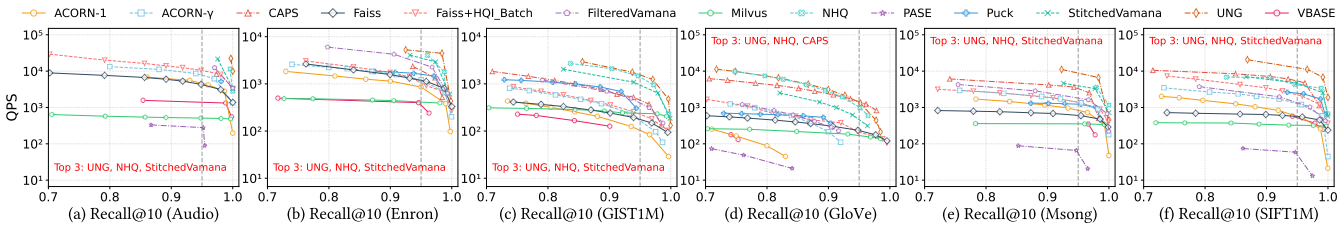



**Figure 4: Single-Attribute Building and Query (single thread)**

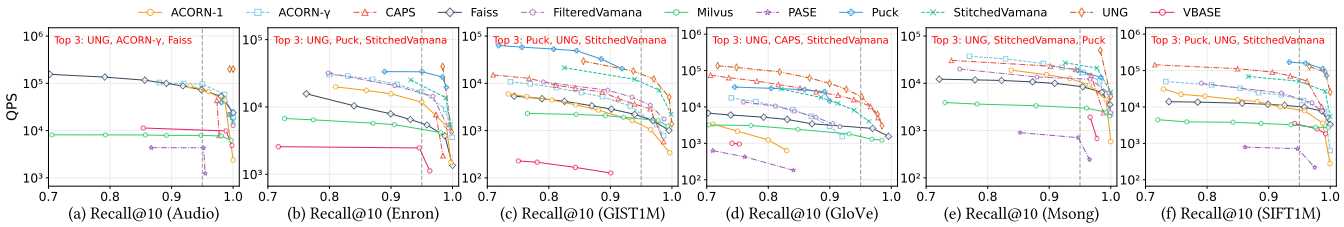



**Figure 5: Single-Attribute Building and Query (16 threads)**

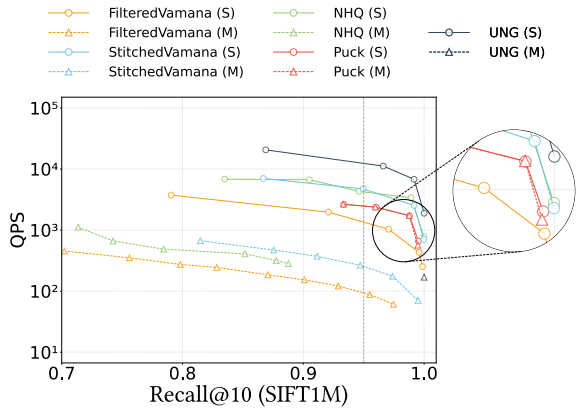



**Figure 6: Effect of Multi-Attribute Index Construction on Single Query Performance. ($S$ denotes single-attribute index construction and single-attribute query; $M$ denotes multi-attribute index construction and single-attribute query)**

number of attributes involved, thus impacting both efficiency and accuracy under multi-attribute filtering.

*4.2.3 Robustness.* In this section, we evaluate the robustness of these algorithms.

***Attribute Distribution.*** To comprehensively evaluate performance across different attribute distributions, we generate base and query attribute sets with four representative distributions —long-tailed, normal, power-law, and uniform—across the six datasets described in Section 4.1.1. We evaluate all algorithms under these distributions on each dataset. The performance differences observed for the same algorithm across different datasets are relatively minor. Thus, due to space constraints, we present only the results on SIFT1M.

As illustrated in Figure 8, the evaluated algorithms exhibit varying degrees of sensitivity to attribute distribution shifts. Specifically, ACORN-1, ACORN-$\gamma$, Milvus, Puck, CAPS, StitchedVamana, and UNG demonstrate strong robustness to such shifts. In contrast, Faiss, FilteredVamana, NHQ, VBASE, and PASE show notable sensitivity.

Among all algorithms, ACORN-$\gamma$ demonstrates the highest robustness. During its index construction, attribute is mainly used for pruning in the lowest layer graph, while the upper-layer hierarchical navigation graphs remain largely unaffected. As a result, changes in attribute distribution have a limited impact on overall performance.

In contrast, VBASE exhibits the weakest robustness. As a post-filtering system, it performs significantly better under uniform distributions but degrades under long-tailed, normal, and power-law distributions. This degradation stems from the low selectivity of certain attributes in the last three distributions, which increases unnecessary distance computations, thus reducing overall efficiency.

***Single-Attribute Selectivity.*** In hybrid queries, different attribute selectivity (AS) levels can significantly affect computational cost and query efficiency, as the algorithm must effectively identify matching data points with specific attributes within a large dataset. AS refers to the proportion of data points in the dataset that share a specific attribute value. For example, an AS of 1% indicates that only 1% of the dataset satisfies the given attribute condition.

To investigate this effect, we evaluate four AS levels (1%, 25%, 50%, and 75%) on the SIFT1M dataset. Using a single-threaded execution environment, we compare the query performance of 13 algorithms under varying AS conditions, as shown in Figure 9.

Experimental results show that UNG achieves the best performance across all AS levels, especially at extremely low AS (1%). This advantage stems from its pre-filtering strategy. Pre-filtering strategy significantly reduces the ANN search space by narrowing down candidates prior to vector computation. In contrast, VBASE performs the worst at 1% AS due to its post-filtering strategy. Post-filtering strategy first retrieves a large number of candidates and then applies filtering, incurring substantial computational overhead.

According to the query performance trends of different algorithms under varying AS levels, we categorize the algorithms into three groups:

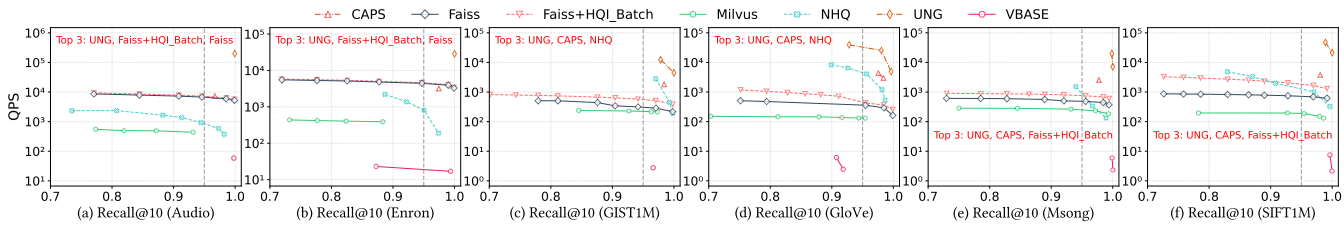



**Figure 7: Multi-attribute build and query (single thread)**

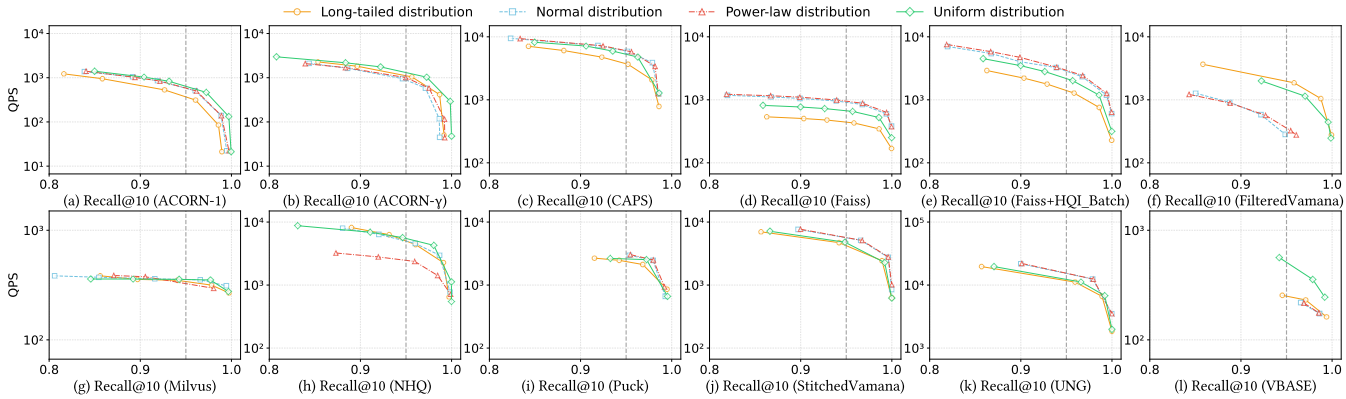



**Figure 8: Effect of attribute distribution on on query performance (single thread)**

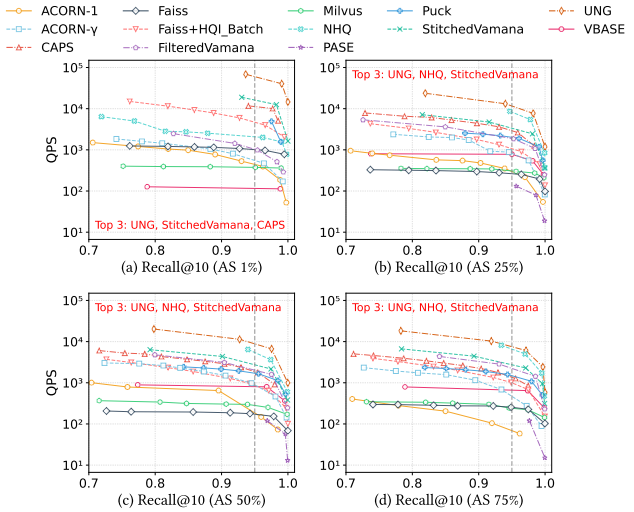



**Figure 9: Effect of Single-Attribute Selectivity on Query**

(1) *Algorithms optimized for low AS levels.* This category includes ACORN-1, CAPS, StitchedVamana, UNG, Faiss, Puck, and Faiss+HQI_Batch, as illustrated in Figure 10a.

ACORN-1 performs well in low AS scenarios because fewer nodes share the same attribute. This increases the average degree of retained nodes and improves graph connectivity. CAPS efficiently skips irrelevant subpartitions, reducing computational overhead. However, as AS increases, it must process more subpartitions and handle a larger dataset. StitchedVamana optimizes local adjacency using independent subgraphs, enabling fast localization at low AS. However, at higher AS levels, it requires broader global exploration. UNG, Faiss, and Puck all use pre-filtering strategies to shrink the search space in low AS conditions, leading to better performance.

Interestingly, Faiss and Faiss+HQI_Batch perform worst at the 50% AS level instead of 75%, which might seem unexpected. This happens because, when AS decreases from 75% to 50%, fewer data points are filtered out. As a result, more clusters must be searched, increasing computational cost.

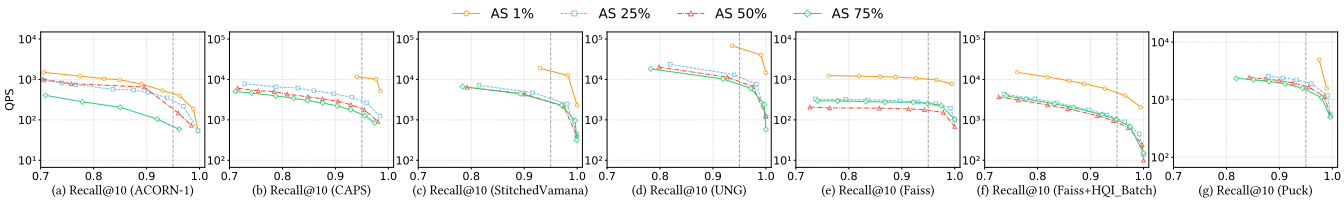
(2) *Algorithms optimized for high AS levels.* This group includes NHQ, FilteredVamana, VBASE, and PASE, as shown in Figure 10b.

At low AS levels, NHQ struggles to locate relevant regions efficiently, leading to excessive computations on non-matching nodes. FilteredVamana also performs poorly in these scenarios. Its dynamic pruning strategy makes it difficult to balance vector distance and attribute relevance, resulting in overly restricted search paths. VBASE relies on post-filtering, which increases computational overhead at low AS. PASE performs poorly at extremely low AS (e.g., 1%), with recall dropping below 0.2. Its fixed candidate set may not contain enough valid results after filtering, leading to incomplete top-$k$ results and significantly reducing recall.

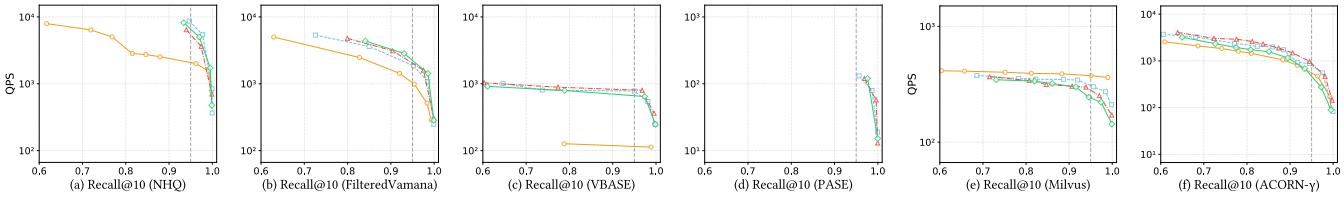
(3) *Algorithms insensitive to AS level.* This group includes Milvus and ACORN-$\gamma$, as illustrated in Figure 10c.

Milvus is mainly limited by system-level communication overhead, such as API latency. As a result, its query performance remains largely unaffected by AS variations. ACORN-$\gamma$ maintains stable performance across different AS levels by using a higher index construction parameter $\gamma$. This ensures a relatively stable average node degree, even after attribute pruning, minimizing the impact of AS changes on query efficiency.

***Different Datasets .*** To evaluate the performance of various methods across different datasets, we conduct experiments on six datasets. Our analysis focuses on performance variations concerning dataset size, LID, and vector dimensionality.




**(a) Algorithms optimized for low AS levels.**




**(b) Algorithms optimized for high AS levels.** **(c) Algorithms insensitive to AS level.**

**Figure 10: Different AS under the same algorithm**

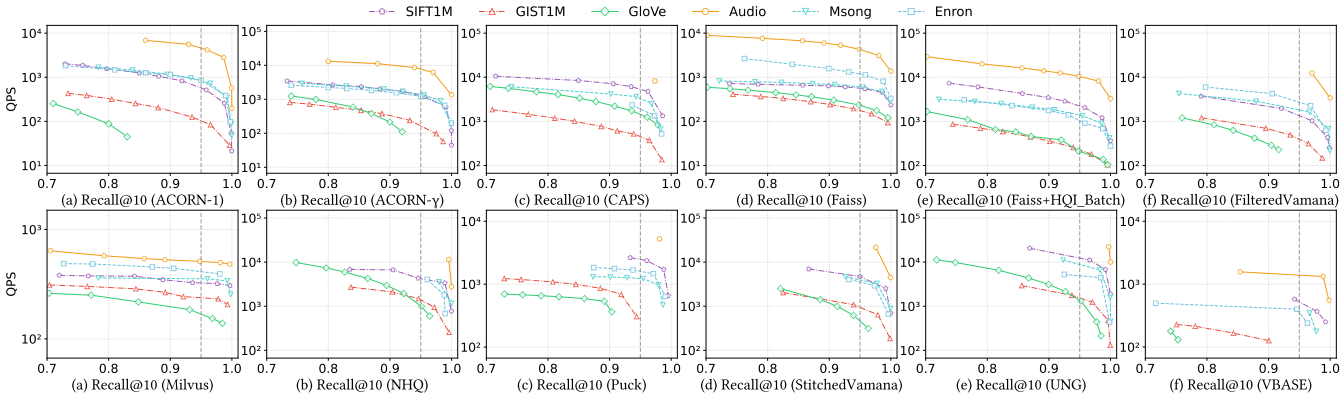



**Figure 11: Effect of dataset on Query**

As shown in Figure 11, all methods achieve their highest QPS and recall on Audio dataset.Because Audio has small scale, low dimensionality, and low LID.In contrast,all methods exhibit the lowest QPS and recall on the GIST1M and GloVe dataset. Because these two dataset have the largest scale and highest LID.

SIFT1M and Msong are similar datasets, with the key difference being that SIFT1M has lower dimensionality. The four algorithms (ACORN,Faiss,PASE, and NHQ) perform similarly on these two datasets, suggesting strong robustness to high dimensional dataset. In contrast, the remaining algorithms exhibit degraded performance on Msong, revealing a lack of adaptability to high dimensional dataset.

Most algorithms struggle to achieve high recall on GloVe,because it has the highest LID. Compare to other algorithms, the four algorithms (UNG, Faiss, CAPS, and Milvus) manage to maintain relatively high recall, demonstrating better adaptability to complex datasets. Notably, although vanilla Faiss performs worse on SIFT1M compared to Enron dataset, Faiss+HQI_Batch outperforms on SIFT1M. Because batch optimization in HQI reduces query overhead and increases throughput.

## 4.3 Range Filtering

Although the libraries and databases mentioned above can also implement range filtering, the results are poor, so we will only compare several range filtering algorithms.

**Table 4: Time and space overhead of range filtering index construction**

| Dataset | Metric | DSG | IRange | SeRF | UNIFY | WinFilter |
|---|---|---|---|---|---|---|
| DEEP | Index time(s) | 4156 | 1128 | **189.89** | 7510 | 32843 |
| | Index size(GB) | 4.6 | 0.92 | **0.80** | 1.43 | 2.41 |
| | Peak memory(GB) | 7.78 | 3.54 | **1.38** | 2.61 | 7.08 |
| YT-Audio | Index time(s) | 15314 | 1039 | **187.20** | 5905 | 30904 |
| | Index size(GB) | 4.3 | **0.7** | 1.04 | 1.55 | 1.32 |
| | Peak memory(GB) | 7.5 | 3.16 | **1.57** | 2.97 | 6.95 |
| WIT | Index time(s) | 29492 | 7068 | **828.12** | 28759 | 94909 |
| | Index size(GB) | 4.2 | **1.5** | 15.34 | 8.71 | 1.59 |
| | Peak memory(GB) | 21.7 | **12.32** | 15.87 | 24.38 | 15.29 |

*4.3.1 Time and Space overhead of index construction.* Similar to the evaluation of *Attribute Filtering* algorithms, we also evaluate three key metrics mentioned above in *Range Filtering* algorithms.

***Index construction time.*** As shown in Table 4, SeRF achieves the fastest index construction time among the five algorithms. It incrementally builds the index in attribute order. It requires only a single pass and adds edges within local graphs. The structure is simple, with no redundant operations.

In contrast, WinFilter has the longest construction time. This is mainly due to its tree-based structure, where each node requires an independent nearest neighbor graph. This leads to significant redundant computations and high resource costs, especially for large datasets.

***Index size.*** Across the three datasets, IRange maintains a relatively small index size. It builds independent local neighbor graphs within each interval and merges indices dynamically during queries. This prevents space explosion. On low-dimensional datasets (Deep, YT-Audio), SeRF compresses the index using sparse vector distributions, keeping it small. However, in the high-dimensional WIT dataset with high LID, vector connectivity increases. This reduces compression efficiency, leading to a sharp rise in index size. DSG maintains a stable index size across datasets but is generally larger. It uses edge structures with interval labels to support streaming insertions and range queries, increasing storage overhead.

***Peak memory usage.*** In terms of peak memory usage, all five algorithms consume less memory on the first two datasets. However, memory usage increases significantly on WIT, likely due to its high dimensionality.

SeRF achieves the lowest memory usage on low-dimensional datasets. Its Segment Graph efficiently compresses sparse vector spaces. However, on high-dimensional datasets with high LID, the "curse of dimensionality" causes distance concentration. As a result, the index structure of SeRF expands rapidly, reducing compression efficiency.

In contrast, IRange achieves the lowest peak memory usage on WIT. It pre-builds graphs only for a limited number of intervals, avoiding explicit index construction for all possible query ranges. Its index structure remains stable in high dimensions without significant graph growth. This effectively controls index size, making IRange more advantageous for high-dimensional datasets.

*4.3.2 Performance Evaluation.* In this experiment, we follow the query range definition adopted by prior work [37]. Specifically, for a given query, if the range covers $n/2^i$ data points—where $n$ is the total dataset size—we define the range ratio as $2^{-i}$. Based on this definition, we evaluate algorithm performance under four query range settings: $2^{-2}$, $2^{-4}$, $2^{-6}$, and $2^{-8}$.

Due to space constraints, Figure 12 reports results for only the largest ($2^{-2}$) and smallest ($2^{-8}$) range ratios. The experimental results show that across all datasets (Deep, YT-Audio, and WIT) and query ranges, WinFilter consistently delivers the best performance. This highlights the robustness of its design, which performs ANN search over only relevant tree nodes and merges results efficiently, making it highly effective for static range query scenarios.

iRange offers a good balance between query efficiency and recall, demonstrating stable performance across both datasets and range sizes. In contrast, UNIFY experiences a notable drop in performance under narrow query ranges, suggesting that its HSIG structure—despite incorporating skip lists for pre-filtering—still has limitations in such scenarios. However, by leveraging HNSW and bitmap-based post-filtering, UNIFY performs competitively under broader range conditions, demonstrating strong scalability.

SeRF exhibits relatively weak overall performance, with a significant drop in recall under small-range query scenarios. This suggests that while its compressed graph structure is space-efficient, it has a substantial negative impact on query performance.

It is important to note that this range query experiment is conducted under a fully static setting. Under such conditions, UNIFY, iRange, SeRF, and WinFilter require the dataset to be sorted by attribute prior to index construction. In contrast, DSG support dynamic index construction with unordered attribute insertions, making them more suitable for streaming or incremental data scenarios. This flexibility highlights their superior extensibility in real-world applications.

*4.3.3 Robustness.* To examine the robustness of RF-ANNS algorithms, we analyze their performance across different datasets. As shown in Figure 13, the performance differences. All methods perform worst on WIT, which has the highest LID. On Deep and YT-Audio, which have similar LID, most algorithms achieve comparable performance. This suggests that the effectiveness of RF-ANN search is mainly influenced by Lid of the dataset. A higher LID inherently increases the difficulty of hybrid query.

However, the DSG algorithm shows a noticeable performance gap between the Deep and YT-Audio datasets, despite their comparable LID. This indicates that DSG is less robust than other methods when facing dataset-dependent variations, possibly due to its sensitivity to data distribution beyond LID alone.

# 5 DISCUSSION

Based on the performance of the algorithms on different experimental scenarios, we discuss our findings as follows.

## 5.1 Recommendations.

**Table 5: Algorithm Recommendation per Scenario**

| Scenario | Attribute Filtering | Range Filtering |
|---|---|---|
| S1: Large-scale Datasets | Puck, UNG | iRange |
| S2: Fast Index Construction | ACORN-1, UNG | SeRF, iRange |
| S3: Query on Hard Datasets | CAPS, UNG | iRange, DSG |
| S4: Query on Easy Datasets | StitchedVamana, NHQ | iRange |
| S5: General-purpose Use | Milvus, Faiss | DSG, UNIFY |
| S6: Resource-constrained Environments | CAPS | SeRF, IRange |
| S7: High-Recall Querying | UNG | WinFilter |

*5.1.1* ***Attribute Filtering.*** We summarize the applicability of different algorithms across various attribute filtering scenarios in Table 5. Puck and UNG excel on large-scale datasets (S1); ACORN-1 and UNG achieve fast index construction (S2); CAPS performs best on hard datasets with high LID (S3), while NHQ, StitchedVamana, and UNG are superior for easy datasets with low LID (S4); CAPS also suits resource-constrained environments due to its compact index and low memory usage (S6); Milvus and Faiss serve as general-purpose solutions (S5); UNG stands out for its high recall, precision, and QPS (S7).

*5.1.2* ***Range Filtering.*** Table 5 also presents the application scenarios for range filtering. iRange demonstrates strong performance on large-scale datasets, combining efficient construction with excellent query capabilities (S1). For fast index construction, SeRF leads with iRange as a close second (S2). Both SeRF and iRange show superior memory efficiency (S6). When handling hard datasets, iRange and DSG are recommended (S3), while iRange also performs well on easy datasets (S4). For general-purpose applications, DSG supports dynamic updates while UNIFY provides flexible filtering strategies (S5). WinFilter emerges as the optimal choice for high-recall static queries (S7).

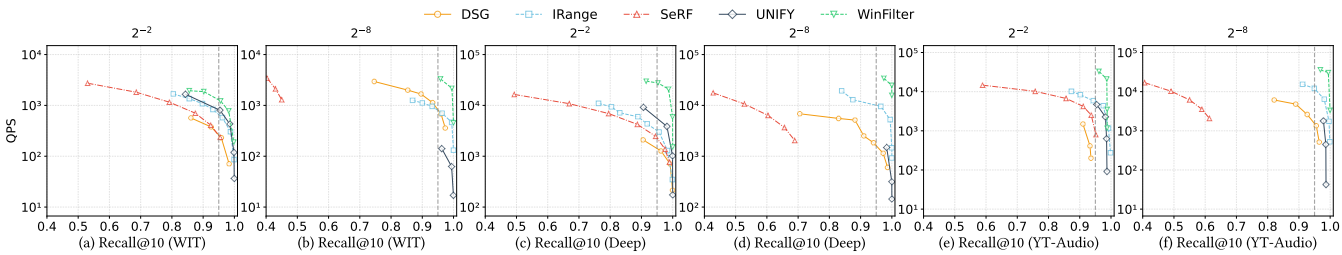



**Figure 12: RF-ANN search**

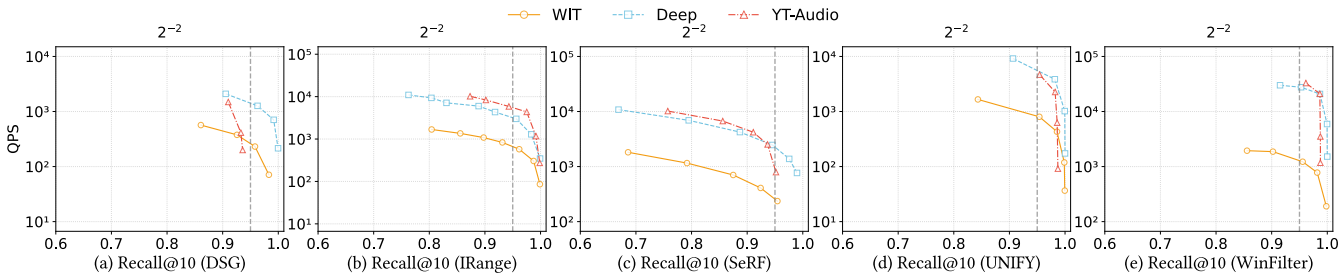



**Figure 13: Dataset Effect on RF-ANN Search**

## 5.2 Challenges

*5.2.1* ***Attribute Filtering.*** Most existing attribute filtering algorithms are graph-based, and they typically outperform other methods in terms of query accuracy and efficiency. However, graph-based algorithms often incur high indexing costs and significant memory overhead. In contrast, IVF algorithms generally consume less memory but suffer from limited performance. Filtered–DiskANN leverages disk-based storage, making it more suitable for extremely large-scale datasets even in memory-constrained environments. Moreover, the index construction phase of graph-based algorithms is computationally intensive. Therefore, exploring GPU-based acceleration for graph construction and vector computation during indexing is promising. It can significantly improve the performance of graph-based methods.

Furthermore, current attribute filtering algorithms offer limited support for complex filtering conditions. For example, Filtered-DiskANN supports single-attribute filtering, and in multi-attribute scenarios, it only supports OR conditions between attributes. NHQ and CAPS impose strict constraints on the number of base and query attributes and only support AND conditions. While UNG is relatively flexible—supporting both AND and OR logic—it still lacks support for arbitrary Boolean expressions (e.g., combinations of AND and OR). The lack of flexible Boolean filtering remains a key challenge, limiting the practicality of hybrid query methods in real-world systems with diverse and complex query requirements.

*5.2.2* ***Range Filtering.*** Except for DSG, most Range Filtered algorithms require the dataset to be pre-sorted before building the index. Moreover, these methods do not support multi-attribute range queries. Enhancing these methods to support an arbitrary number of range-filterable attributes remains an open research challenge.

Our experiments show that graph compression methods often lead to poor query performance in RF-ANN search. To address this problem, subsequent algorithms usually optimize the index processing at the segment tree nodes. UNIFY newly proposes segment graphs and integrates multiple data structures to process queries. Although UNIFY performs slightly worse on small-range queries, its proposed index structure broadens the research direction.

It is also worth noting that range filtering and attribute filtering are both forms of hybrid queries. However, aside from vector databases and ACORN, few existing algorithms support both modalities simultaneously. Designing unified frameworks that simultaneously support both is an important research frontier.

Lastly, our findings show that the distribution and cardinality of attributes, as well as dataset properties (e.g., dimensionality and LID), significantly impact the performance of attribute-filtering methods. Enhancing the robustness of these algorithms under diverse data conditions is another key challenge that warrants further investigation.

## 6 CONCLUSION

In this paper, we evaluate various hybrid query methods, including algorithms proposed in research papers, vector databases, and vector libraries. For attribute filtering algorithms, we design and conduct a series of experiments to comprehensively assess their overall performance. We then perform experiments on six real-world datasets to analyze the effectiveness of attribute filtering algorithms in depth. Additionally, we compare existing range filtering algorithms using three range query datasets. Finally, we provide a detailed analysis of the experimental results, summarize key findings, and highlight areas for improvement. Our study not only validates previous research but also offers insights for future work.

# REFERENCES


[1] [n.d.]. WIT: Wikipedia-based Image Text Dataset. https://github.com/google-research-datasets/wit

[2] [n.d.]. YouTube-8M: A Large-Scale Video Classification Benchmark. https://research.google.com/youtube8m/download.html

[3] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2015. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 29–38.

[4] Anon. 2010. Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr/.

[5] Anon. 2011. Million Song Dataset Benchmarks. http://www.ifs.tuwien.ac.at/mir/msd/.

[6] Anon. unknown. TIMIT Audio. https://www.cs.princeton.edu/cass/demos.htm.

[7] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*. 573–582.

[8] Martin Aumüller and Matteo Ceccarello. 2019. Benchmarking nearest neighbor search: influence of local intrinsic dimensionality and result diversity in real-world datasets. In *Proceedings of the 1st Workshop on Evaluation and Experimental Design in Data Mining and Machine Learningco-located with SIAM International Conference on Data Mining (SDM 2019)*, Vol. 2436. CEUR-WS, 14–23.

[9] Baidu. 2023. Puck: A High-Performance ANN Search Framework. https://github.com/baidu/puck.

[10] Dmitry Baranchuk and Artem Babenko. [n.d.]. Benchmarks for Billion-Scale Similarity Search. https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search DEEP dataset description and download information.

[11] Jeffrey S Beis and David G Lowe. 1997. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. *CVPR* (1997), 1000–1006.

[12] Yuzheng Cai et al. 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.

[13] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. SPTAG: A library for fast approximate nearest neighbor search. https://github.com/microsoft/SPTAG.

[14] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.

[15] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[16] 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.

[17] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[18] 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).

[19] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.

[20] Siddharth Gollapudi et al. 2023. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. *Proceedings of the ACM Web Conference* (2023).

[21] Gaurav Gupta et al. 2023. CAPS: A practical partition index for filtered similarity search. *arXiv preprint arXiv:2308.15014* (2023).

[22] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and Thomas Legrand. 2019. Applying Deep Learning to Airbnb Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 1927–1935. https://doi.org/10.1145/3292500.3330658

[23] 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.

[24] 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.

[25] Masajiro Iwasaki. 2016. Pruned bi-directed k-nearest neighbor graph for proximity search. In *International Conference on Similarity Search and Applications*. Springer, 20–33.

[26] 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).

[27] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

[28] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. https://doi.org/10.1109/TPAMI.2010.57

[29] J. Li, H. Liu, C. Gui, J. Chen, Z. Ni, N. Wang, and Y. 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*. 9–16.

[30] 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.

[31] 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).

[32] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related Pins at Pinterest: The Evolution of a Real-World Recommender System. In *Proceedings of the 26th International Conference on World Wide Web Companion* (Perth, Australia) *(WWW '17 Companion)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 583–592. https://doi.org/10.1145/3041021.3054202

[33] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, et al. 2022. Monolith: real time recommendation system with collisionless embedding table. *arXiv preprint arXiv:2209.07663* (2022).

[34] 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.

[35] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE TPAMI* 42, 4 (2018), 824–836.

[36] 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.

[37] 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. 1, 2, Article 197 (June 2023), 25 pages. https://doi.org/10.1145/3589777

[38] 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.

[39] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. Pinnersage: Multi-modal user embedding framework for recommendations at pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2311–2320.

[40] 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.

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

[42] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2015. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/.

[43] PostgreSQL Global Development Group. 2020. PostgreSQL. https://github.com/postgres/postgres/tree/REL_13_4.

[44] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Dobson Manohar, Dmitry Baranchuk, Edo Liberty, Frank Liu, Ben Landrum, et al. 2024. Results of the Big ANN: NeurIPS'23 competition. *arXiv preprint arXiv:2409.17424* (2024).

[45] Russell Stewart, Christopher Manning, and Jeffrey Pennington. 2015. Enron Email Dataset. https://www.cs.cmu.edu/~./enron/.

[46] Zilliz Tech. 2024. VectorDBBench: A Benchmark for Vector Databases. https://github.com/zilliztech/VectorDBBench.

[47] Yao Tian, Ziyang Yue, Ruiyuan Zhang, Xi Zhao, Bolong Zheng, and Xiaofang Zhou. 2023. Approximate Nearest Neighbor Search in High Dimensional Vector Databases: Current Research and Future Directions. *IEEE Data Eng. Bull.* (2023).

[48] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 839–848.

[49] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-nn graph construction for visual descriptors. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1106–1113.

[50] 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.

[51] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2023. An efficient and robust framework for approximate nearest neighbor search with attribute constraint. *Advances in Neural Information Processing Systems* 36 (2023), 15738–15751.

[52] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2023. An efficient and robust framework for approximate nearest neighbor search with attribute constraint. *Advances in Neural Information Processing Systems* 36 (2023), 15738–15751.

[53] 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).

[54] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, 194–205.

[55] 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. https://doi.org/10.14778/3415478.3415541

[56] Yuexuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S. Jensen. 2024. iRangeGraph: Improvising Range-dedicated Graphs for Range-filtering Nearest Neighbor Search. 2, 6, Article 239 (Dec. 2024), 26 pages. https://doi.org/10.1145/3698814

[57] 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.

[58] 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.

[59] 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.