An Experimental Evaluation of Hybrid Vector Search (EA&B)

Jiaxu Zhu
Huazhong University of Science and Technology

Jiayu Yuan
Huazhong University of Science and Technology

Kaiwen Yang
Huazhong University of Science and Technology

Xiaobao Chen
Huazhong University of Science and Technology

Shihuan Yu
Huazhong University of Science and Technology

Hongchang Lv
Huazhong University of Science and Technology

Yan Li
Huazhong University of Science and Technology

Bolong Zheng
Huazhong University of Science and Technology

ABSTRACT

Recent studies demonstrate the significant practical value of hybrid queries, which integrate vector search with structured filters (e.g., attribute and range filtering) for refined retrieval. However, current evaluations lack unified benchmarking standards and systematic assessment methodologies. Existing studies not only fail to cover mainstream algorithms but also omit systematic comparisons or in-depth analyses of different methods. To address this issue, we design a complete evaluation framework for hybrid queries. Our study introduces 15 hybrid query algorithms and systematically classifies them based on multiple dimensions such as index organization and filtering strategy, providing a reference for the categorization of hybrid queries. In the experiments, for attribute filtering, we construct standarded attribute sets, enabling a unified comparison of algorithms in terms of index construction efficiency, query performance, and robustness. For range filtering, we systematically evaluate algorithm performance across these 3 metrics through controlled variation of query ranges. Additionally, we conduct an in-depth analysis of the experimental results based on the underlying principles of the algorithms. The extensive experimental results reveal the strengths and weaknesses of each algorithm. Based on these findings, we develop a set of practical guidelines for algorithm selection, offering reliable references for different application scenarios. Furthermore, we identify potential directions for improvement to address the current limitations of these algorithms.

PVLDB Reference Format:
Jiaxu Zhu, Jiayu Yuan, Kaiwen Yang, Xiaobao Chen, Shihuan Yu, Hongchang Lv, Yan Li, and Bolong Zheng. An Experimental Evaluation of Hybrid Vector Search (EA&B). 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-Experiment.

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

1 INTRODUCTION

Nearest Neighbor (NN) search [15] aims to find the closest vector in a given space and serves as a fundamental algorithm in vector retrieval, with widespread applications in recommendation systems [32] and image retrieval [45]. However, with the exponential growth of data volume and the increasing dimensionality of vectors [50], traditional NN search methods struggle to meet the demands of real-time search. To address this issue, researchers turn to more efficient approaches - Approximate NN search [7, 11, 20, 34]. ANN algorithms significantly improve search efficiency by constructing effective indexing structures, albeit at the cost of reduced accuracy.

Nevertheless, as application scenarios grow increasingly complex, simple ANN search can no longer satisfy all practical needs. For instance, Figure 1 illustrates a case where users on e-commerce platforms search for clothing items by retrieving visually similar products based on an image. Additionally, users may impose further requirements such as price, color, or brand preferences. Such scenarios necessitate a retrieval system capable of simultaneously addressing vector similarity (e.g., product image) and attribute constraints (e.g., brand name) [44]. When the constraint involves a specific attribute value, this problem refers to Attribute Filtering Approximate Nearest Neighbor (AF-ANN) search [21, 47] . For example, a user may seek a green piece of clothing. If the constraint involves a range condition, the problem is known as Range Filtering Approximate Nearest Neighbor (RF-ANN) search [52, 55]. For instance, filtering clothes priced between 100 and 200. To meet these application demands, hybrid query [29, 51] techniques emerge. Hybrid queries integrate vector retrieval with conditional filtering, optimizing their interaction to significantly enhance efficiency and flexibility in complex query scenarios.

1.1 Motivation

In recent years, hybrid query algorithms develop rapidly, giving rise to numerous attribute filtering [26, 47] and range filtering algorithms [52, 55]. In practical applications, the performance of hybrid query algorithms is influenced not only by unstructured data but also closely related to structured data. For attribute filtering algorithms, factors such as the number of base attributes, the number of query attributes, attribute distribution [12], and attribute selectivity [37] significantly impact algorithm performance. As for range

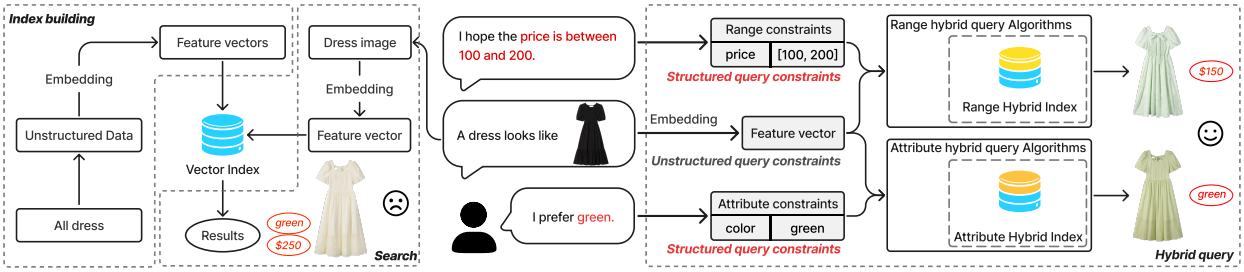


Figure 1: Hybrid query example

filtering algorithms, the size of the query range and the characteristics of different datasets also play an important role in determining algorithm performance.

Despite BigANN 2023 [41] evaluates the performance of some algorithms, it presents several notable limitations: 1) The evaluation covers a limited number of algorithms and fails to comprehensively include mainstream methods. For instance, attribute filtering algorithms such as NHQ and UNG are not included in the evaluation. 2) It focuses only on single-attribute and dual-attribute query scenarios, neglecting the more complex demands of multi-attribute queries. In real-world applications, an object typically involves multiple attributes.For example, in e-commerce scenarios, users often specify multiple attributes (e.g., color, brand, price range) simultaneously when searching. 3) It does not evaluate algorithm performance under different attribute distributions. However, variations in attribute distribution may directly affect the filtering strategies and index construction efficiency of hybrid query algorithms.

Moreover, aside from BigANN, there is currently a lack of systematic evaluation specifically targeting hybrid query algorithms. To fill this gap, we conduct a comprehensive experimental evaluation of hybrid query algorithms, analyzing the index construction costs and query efficiency across different scenarios. Additionally, we further assess the robustness of each algorithm under varying conditions.

1.2 Our Contributions

Our study focuses on the problem of ANN search in hybrid query scenarios and provides a comprehensive review and evaluation of existing algorithms and systems. The main contributions are summarized in the following 4 aspects.

(1) **Systematic Classification and Overview.** We systematically classify 10 representative attribute filtering algorithms from multiple dimensions, including index organization, filtering strategy, Boolean logic support, and index construction methods. Additionally, we survey range filtering algorithms and provide an overview of widely used vector retrieval libraries (e.g., Faiss) and vector databases (e.g., Milvus, PASE, VBASE). These efforts provide a unified reference framework for future study.

(2) **Enhancing Datasets and Experimental Settings for Fair Evaluation.** To address the lack of standardized benchmarks in attribute filtering research, we enrich commonly used datasets by generating attribute values tailored to real-world scenarios. Furthermore, we design comprehensive experimental settings that reflect diverse application requirements, including varying attribute distributions, selectivity levels, and query conditions. These enhancements provide a unified evaluation framework that supports fair, consistent, and reproducible comparisons across different algorithms, laying a solid foundation for future research in hybrid query processing.

(3) **Evaluation of Attribute Filtering Algorithms.** We conduct a systematic evaluation of 13 attribute filtering algorithms on 7 real-world datasets. By analyzing performance under varying numbers of attributes, we reveal the strengths and weaknesses of each method. We further examine their behavior under different attribute distributions and selectivity conditions to assess robustness and adaptability in complex query scenarios. Evaluation metrics include index construction time, index size, peak memory usage, QPS, and search accuracy.

(4) **Evaluation of Range Filtering Algorithms.** We benchmark 5 mainstream range filtering algorithms on 3 large-scale datasets, using varying query range settings in the experiments. The experimental results demonstrate the performance of these algorithms in terms of index construction efficiency, storage overhead, and query performance. Additionally, we conduct an in-depth analysis of how index organization impacts algorithm performance, providing valuable insights for algorithm design.

(5) **Recommendations and Challenges.** Based on the experimental results, we provide algorithm selection recommendations for common application scenarios and highlight key challenges in the field of hybrid queries. These challenges include limited Boolean logic support, the lack of multi-attribute range filtering capabilities, the high indexing costs of graph-based methods, and the sensitivity of algorithms to data distribution. Currently, few methods simultaneously support both attribute filtering and range filtering, pointing to potential directions for future research.

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 studies 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, and $\hat{R}$ 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 2 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 $D_s = \{v_i \mid (v_i, s_i) \in D \wedge q_s \subseteq s_i\}$.

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.

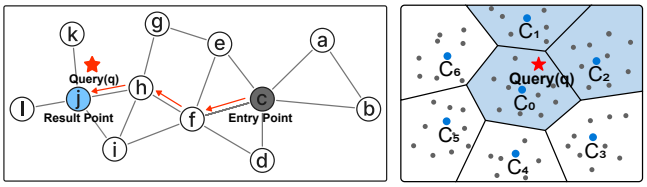
2.2 Index Organization in Hybrid ANN Search

Current hybrid query methods mainly adopt graph-based [17, 19, 23, 25, 33] or Inverted File Index (IVF)-based [27] index organization. We briefly introduce the fundamental principles of these 2 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 5 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., 1NN): 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.



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

Figure 2: Graph index and IVF index

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.

3 OVERVIEW OF HYBRID QUERY ALGORITHMS

We provide an overview and analysis of 15 hybrid query methods, all drawn from recent studies, and collectively referred to as algorithms in the following. Specifically, Section 3.1 describes methods for attribute filtering, Section 3.2 discusses those for range filtering, and Section 3.3 introduces vector libraries and databases that support both attribute filtering and range filtering.

3.1 Attribute Filtering Algorithms

We summarize 6 attribute filtering algorithms and provide an intuitive comparison in Table 1.

NHQ [47]. 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 [21]. 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 similarity and attribute constraints. Moreover, Filtered-DiskANN supports SSD-based storage, improving scalability. Filtered-DiskANN proposes 2 indexes: 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

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	Y	N	Y	Y
ACORN	Graph	B	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.

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.

CAPS [22]. Different from the above graph-based simultaneous filtering algorithms, CAPS is the first simultaneous filtering algorithm based on spatial partitioning. 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 [37]. Above simultaneous filtering algorithms struggle with large-scale, unbounded, or unknown predicate sets. ACORN addresses this by introducing a predicate-agnostic indexing framework. Built upon Hierarchical Navigable Small World (HNSW) [35], ACORN supports high-cardinality and unrestricted predicates, overcoming limitations of methods confined to small-scale equality filters. It constructs a denser hierarchical graph by expanding neighborhoods and prunes lower-level edges to control index size. During querying, ACORN eliminates unnecessary distance computations by filtering out nodes violating attribute constraints, effectively maintaining a nearest neighbor graph over valid nodes only. ACORN includes two indexes: 1) **ACORN-γ**, which expands neighbor lists during construction, trading memory for higher performance; 2) **ACORN-1**, which extends neighbor lists via second-hop neighbors during search, reducing index size with minor performance loss.

UNG [12]. The above simultaneous filtering algorithms may not guarantee the completeness of the results and perform poorly when the attribute selectivity is low. UNG is proposed to overcome these 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. UNG supports Boolean filtering with 2 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.

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.

Puck [10]. While simultaneous filtering algorithms perform 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 5 representative range filtering algorithms and present a comparative overview in Table 2.

SeRF [55]. 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 [48]. Unlike SeRF, which focuses on graph compression, WinFilter proposes a structural partitioning framework called the β-Window Search Tree (β-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 [52]. 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 [38]. Most RF-ANN search methods, such as iRange and WinFilter, are designed for static datasets. SeRF allows incremental insertion but requires ordered attributes. DSG introduces the first dynamic RF-ANN framework supporting data insertion with unordered attributes while maintaining efficient range filtering. DSG relies on two data structures: 1) A rectangle tree partitions query space into rectangular regions, each corresponding to a group of queries sharing nearest neighbors. 2) A dynamic segment graph is a neighbor graph where each edge is annotated with its valid attribute range. With these structures, only few regions need updates when inserting new data. During the query process, the system considers only edges valid for current attribute range, boosting 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-filtering pruning. UNIFY supports 3 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 the following 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 [18]. Faiss is a library designed for efficient similarity search. It supports various indexing structures, including IVF, Product Quantization (PQ) [27], 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 process, Faiss excludes vectors based on the selector, enabling efficient integration of attribute filtering with ANN search. In addition, we adopt the batch optimization strategy of HQI [36] 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.

Table 3: Datasets

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
Text2Image	200	10,000,000	10,000	59	Text2Image

PASE [53]. PASE is a vector indexing plugin for the PostgreSQL database [40], supporting two index types: IVF_Flat [28] and HNSW. By leveraging database capabilities, PASE enables both attribute filtering and range filtering. We focus on the post-filtering strategy based on HNSW, which facilitate comparison with other methods. 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 [54]. 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 [46]. Unlike PASE and VBASE, which are database extensions, Milvus is an 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 7 real-world datasets widely adopted in existing literature: Msong [5], Audio [6], SIFT1M [4], GIST1M [4], GloVe [39], Enron [42] and Text2Image [16], covering a diverse range of domains. We independently generate attribute for each dataset.

For range filtering queries, we evaluate 3 widely used real-world datasets: Deep [9], YT-Audio [2], and WIT [1]. Following prior work [38], 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.

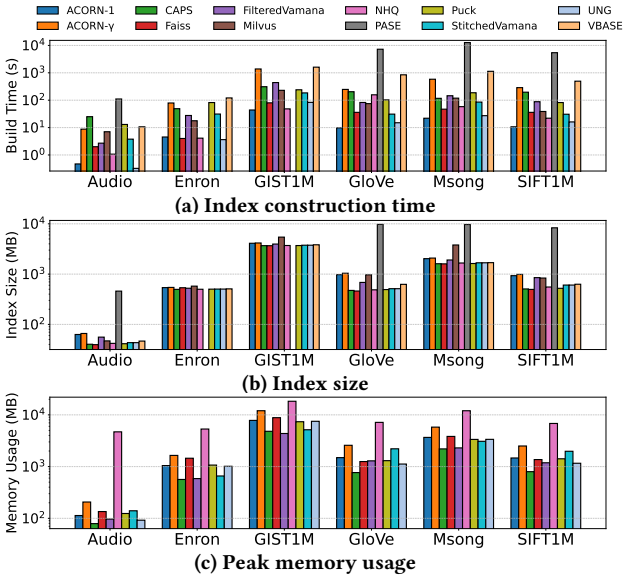


Figure 3: Time and space overhead of index construction

4.1.2 Evaluation Metrics. To comprehensively assess the overall performance of different algorithms, we adopt a multi-dimensional quantitative evaluation framework [49], which includes the following 5 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 index construction.

4.1.3 Parameter Settings. For all evaluated algorithms, we adopt the parameter settings recommended in the original study 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 [43], to better reflect their real-world performance.

4.2 Attribute Filtering

4.2.1 Time and Space overhead of index construction. To investigate the performance of different attribute filtering algorithms during index construction, we evaluate 3 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 4 datasets, PASE has the longest index construction time due to its single-thread indexing support. Additionally, VBASE also supports only single-thread 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-γ incurs the highest index construction time, as it expands neighbor lists and evaluates a large candidate pool, leading to increased computational overhead.

Index size. Traditional graph-based methods often only evaluate the size of the generated graph index and do not include the original data, while the index generated by the IVF-based method includes the original data. In order to unify the comparison standard, we include both the index structure generated by the algorithm and the original data when evaluating the index size. As shown in the Figure 3b, PASE exhibits significantly larger index sizes across all supported datasets. This overhead stems primarily from its extensive metadata requirements. The index sizes of other methods show moderate variation. Among these, Faiss generates slightly smaller indexes due to its efficient index design.

Peak memory usage. As shown in Figure 3c, on small-scale datasets, all methods except NHQ exhibit low memory usage. This is because NHQ adopts the kgraph approach for index construction, which requires loading the entire graph data into memory and maintaining a complete graph structure, leading to relatively high memory overhead. On large-scale datasets, CAPS exhibits slightly lower memory usage compared to other algorithms. Due to their special characteristics, databases are 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 3 main scenarios.

Single-Attribute Building and Single-Attribute Query. In this scenario, we construct the index using an attribute and apply filtering conditions on the same attribute.

Single-Thread 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.

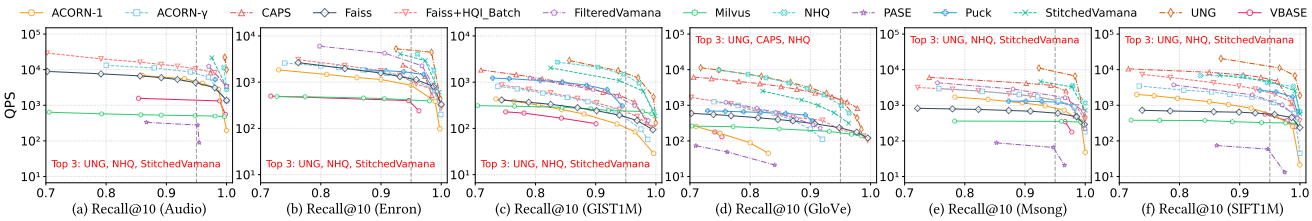


Figure 4: Single-Attribute Building and Query (single thread)

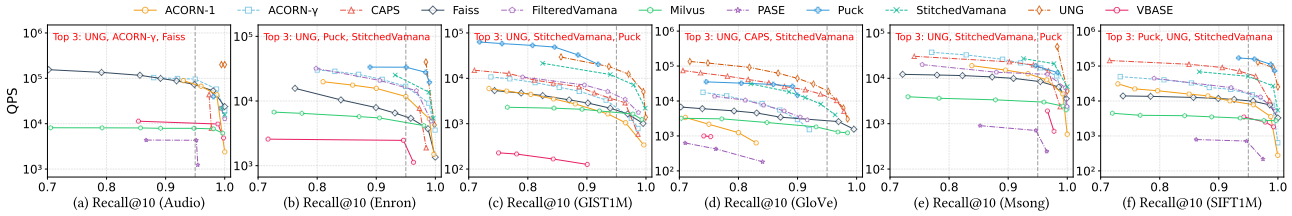


Figure 5: Single-Attribute Building and Query (16 threads)

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 original Faiss, indicating that batch querying via HQI can effectively enhance query performance.

Multi-Thread 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-threads 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 γ 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 8 attributes but applies filtering condition on only 1 attribute during the query process. The original NHQ implementation does not support this scenario. We modified the code to enable this experiment.

Faiss offers an ID filtering mechanism, it operates solely during query execution and does not influence index building. ACORN-1 and ACORN-γ are specifically designed for single-attribute indexing. In database systems (VBASE, PASE, Milvus), when the system

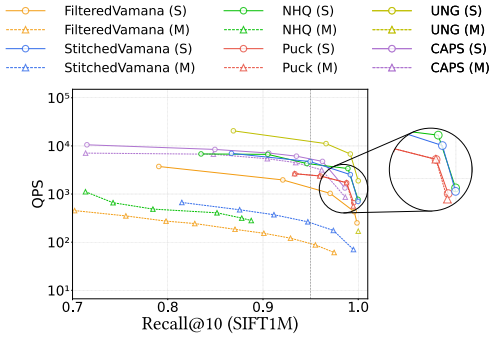


Figure 6: Effect of Multi-Attribute Index Construction on Single Query Performance. (*S* denotes single-attribute index construction and single-attribute query; *M* denotes multi-attribute index construction and single-attribute query)

builds both B+ tree and vector indexes, queries typically utilize only the B+ tree. 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, CAPS shows a slight performance decline due to the increased query overhead caused by excessive attribute-based grouping. Compared with CAPS, the performance of StitchedVamana, FilteredVamana, NHQ, and UNG drops significantly, with the following reasons: 1) FilteredVamana increases the complexity introduced by retaining attribute during index construction. 2) The complexity of StitchedVamana increases because data points may duplicate across multiple subgraphs. 3) The fusion distance of NHQ exhibits sensitivity to the number of attributes, which degrades its effectiveness. 4) UNG has poor performance because large number of entry points that need to be traversed. It is difficult for a graph with multiple entries to have good connectivity, degenerating into a situation closer to inverted retrieval, and the effect may be greatly impaired.

Performance of Algorithms under Multi-Attribute Building (M vs. M). As shown in Figure 6, IVF-based methods (Puck and CAPS) significantly outperform graph-based methods. The reason for Puck's superior performance is that it filters out irrelevant results during the retrieval process rather than searching for more candidates.

The performance of CPAS is as good as that of Puck. The difference between them is that CPAS further groups the results by attributes within clusters. In contrast, FilteredVamana performs the worst because its multi-attribute indexing forces neighbor selection to cover multiple attributes, thereby reducing the efficiency of single-attribute queries. This results in excessive traversal of irrelevant nodes, 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 3 uniformly distributed attributes and apply filtering conditions on all 3 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 medium-level 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 fusion distance calculations. The fusion distance computation is highly sensitive to the number of attributes involved, thus impacting both efficiency and accuracy under multi-attribute filtering.

4.2.3 Robustness. In the following, 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 4 representative distributions—long-tailed, normal, power-law, and uniform—across the 6 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-γ, 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-γ demonstrates the highest robustness. During 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 3 distributions, which increases unnecessary distance computations, thus reducing overall efficiency.

Single-Attribute Selectivity. In hybrid queries, varying levels of attribute selectivity (AS) can significantly impact computational cost and query efficiency, as the algorithm must efficiently identify data points matching specific attributes within large datasets. AS is defined as the proportion of data points sharing a given attribute value. For example, AS of 1% indicates that only 1% of the dataset meets the given attribute.

To investigate this effect, we evaluate 4 AS levels (1%, 25%, 50%, and 75%) on the SIFT1M dataset. Using a single-thread 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 3 groups:

(1) *Algorithms optimized for low AS levels.* This group includes ACORN-1, Faiss+HQI_Batch, StitchedVamana, UNG, Faiss, Puck, and CAPS, 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

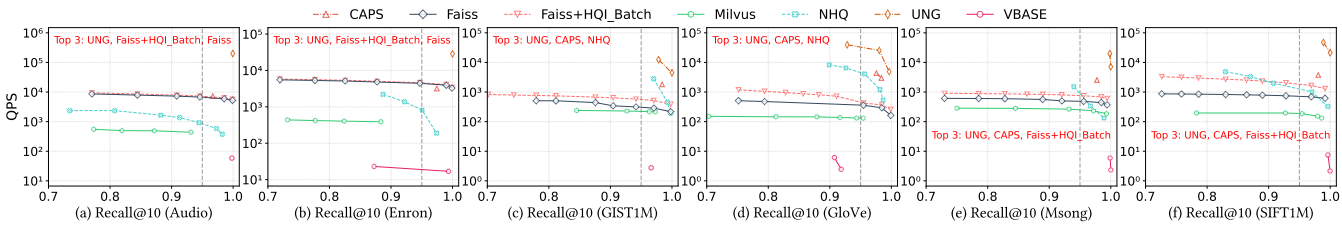


Figure 7: Multi-attribute build and query (single thread)

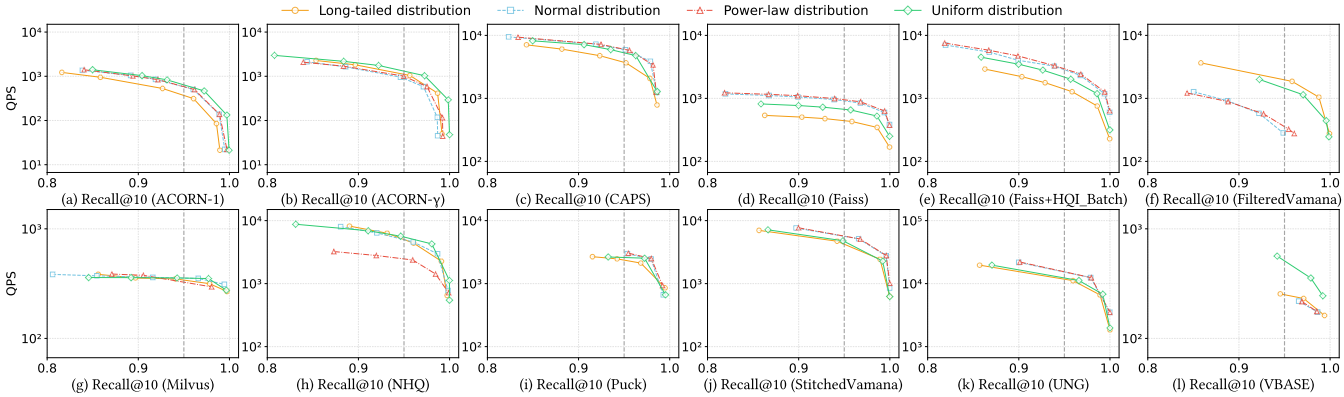


Figure 8: Effect of attribute distribution on query performance (single thread)

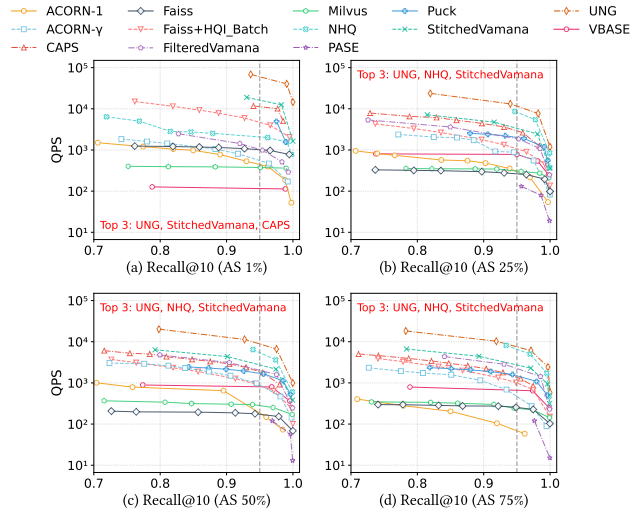


Figure 9: Effect of Single-Attribute Selectivity on Query

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-γ, 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-γ maintains stable performance across different AS levels by using a higher index construction parameter γ. 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 7 datasets, including one dataset for out-of-distribution (OOD) queries. Our analysis focuses on performance variations concerning dataset size, LID, and vector dimensionality.

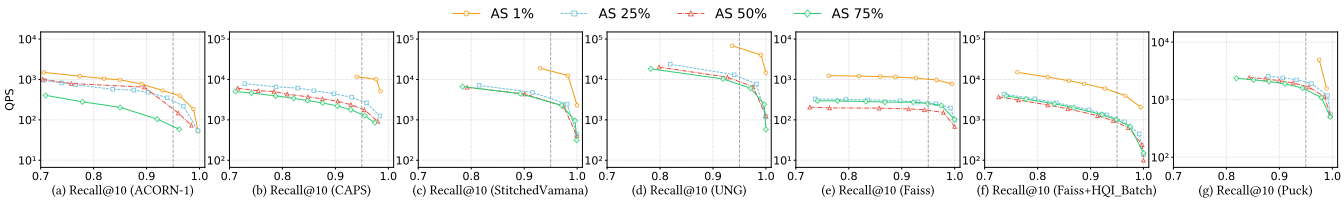
As shown in Figure 11, all methods achieve their highest QPS and recall on Audio dataset. Because Audio has small scale, low dimensionality, and low LID. In contrast, all methods exhibit the lowest QPS and recall on the GIST1M and GloVe dataset besides the OOD dataset Text2Image. Because these two datasets have the largest scale and highest LID.

SIFT1M and Msong are similar datasets, with the key difference being that SIFT1M has lower dimensionality. The 4 algorithms (ACORN, Faiss, PASE, and NHQ) perform similarly on these two datasets, suggesting strong robustness to high dimensional datasets. In contrast, the remaining algorithms exhibit degraded performance on Msong, revealing a lack of adaptability to high dimensional datasets.

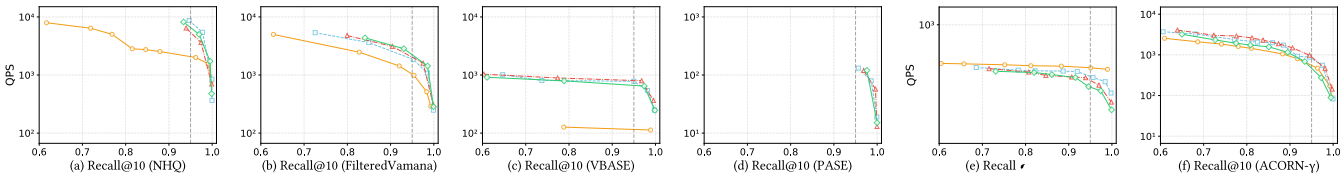
Most algorithms struggle to achieve high recall on GloVe, because it has the highest LID. Compare to other algorithms, the 4 algorithms (UNG, Faiss, CAPS, and Milvus) manage to maintain relatively high recall, demonstrating better adaptability to complex datasets. Notably, although vanilla Faiss performs worse on SIFT1M compared to Enron dataset, Faiss+HQI_Batch outperforms on SIFT1M. Because batch optimization in HQI reduces query overhead and increases throughput.

Most algorithms exhibit the worst performance on the OOD dataset Text2Image. However, ACORN, UNG, and FilteredVamana show relatively stable performance on Text2Image. Their performance drops less compared to in-distribution (ID) datasets such as GIST1M and GloVe, indicating better adaptability to OOD queries.

4.3 Range Filtering



(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

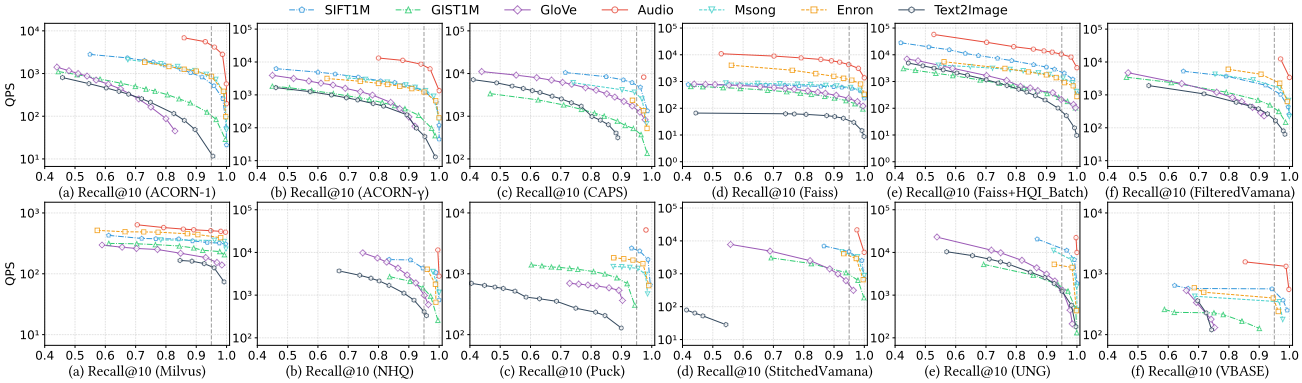
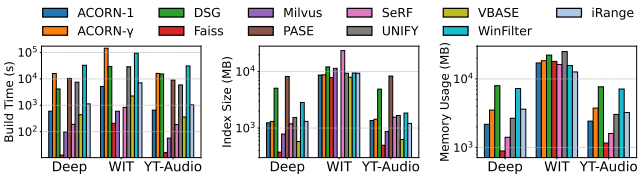


Figure 11: Effect of dataset on Query



(a) Construction time **(b) Index size** **(c) Peak memory usage**

Figure 12: Time and space overhead of index construction

4.3.1 Time and Space overhead of index construction. Similar to the evaluation of *Attribute Filtering* algorithms, we also evaluate 3 key metrics mentioned above in *Range Filtering* algorithms.

Index construction time. Figure 12a shows the single-threaded index construction time. Faiss builds the index fastest. As Faiss obtains subsamples through sampling and calculates the cluster centroid, the computational complexity is significantly reduced.

WinFilter and ACORN-γ take the longest. WinFilter relies on a tree structure, requiring separate nearest-neighbor graphs per node. This causes redundant computation and high costs, especially for large datasets. ACORN-γ expands the neighbor list of each node to provide more candidate paths, which increases both computation and storage costs. It also prunes distant neighbors to save storage space. However, this introduces additional overhead and slows down the construction process.

Index size. As shown in Figure 12b, Among all algorithms, Faiss has the smallest index size. Because IVF only adds centroid data and partition information to the vector dataset, the IVF index is only slightly larger than the vector dataset. On low-dimensional datasets (Deep, YT-Audio), PASE requires the most storage. Its implementation likely wastes space due to alignment padding and inefficient encoding (e.g., Base64).

SeRF maintains a compact index for low-dimensional data. However, for high-dimensional dataset (WIT), SeRF stores more edges and dynamic range data, significantly increasing its size.

Peak memory usage. As Figure 12c shows, Faiss uses the least memory on low-dimensional datasets. It stores minimal temporary data (e.g., vectors, centroids, distances) during index construction.

On low-dimensional datasets, DSG and WinFilter exhibit higher peak memory usage compared to other algorithms. This suggests that their index structures suffer from severe structural redundancy and are difficult to compress in low-dimensional scenarios.

As Figure 12c shows, Faiss uses the least memory on low-dimensional datasets (Deep, YT-Audio). It stores minimal temporary data (e.g., vectors, centroids, distances) during index construction.

For high-dimensional dataset WIT, iRange achieves the lowest peak memory usage. 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. In contrast, Unify exhibits the highest memory consumption, indicating that its index construction process is highly sensitive to the dimensionality of the data.

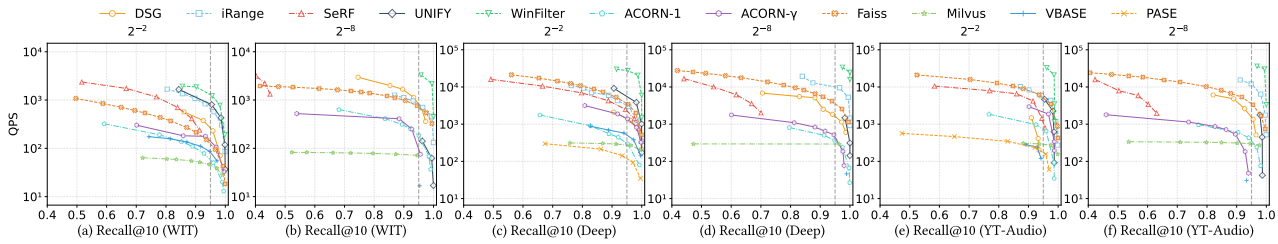


Figure 13: RF-ANN search

4.3.2 Performance Evaluation. In this experiment, we follow the query range definition adopted by prior work [36]. 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 4 query range settings: 2^{-2}, 2^{-4}, 2^{-6}, and 2^{-8}.

Due to space constraints, Figure 13 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.

ACORN performs similarly on both large and small range queries, but its overall performance is worse than the other methods. Unlike other range filtering algorithms, ACORN is not designed with specialized data structures (such as the β-WST in WinFilter, the rectangle tree and dynamic segment graph in SeRF, or the HSIG in UNIFY) to achieve efficient range filtering. In fact, the graph structure used by ACORN for attribute filtering is the same as that used for range filtering.

For third-party libraries and databases, Figure 13 shows that Faiss performs well in range filtering and even outperforms some specialized range query algorithms in certain cases. Faiss does not include optimizations specifically tailored for range filtering scenarios; its query mechanism is identical to that used in tag filtering scenarios. However, since attributes are pre-sorted in range filtering scenarios, the filtering overhead is minimal, leading to improved query performance. Similar to the selectivity observed in attribute filtering, Faiss performs better in small-range filtering scenarios because it searches over the filtered vectors. With a smaller range, fewer points remain after filtering, resulting in lower computational cost. In addition, the performance of Faiss is influenced by data dimensionality; it tends to work better on low-dimensional datasets.

Vector databases also show relatively poor performance on range queries. Among them, Milvus performs more balanced across both small-range and large-range queries, even surpassing ACORN under high recall settings. PASE performs poorly on small-range queries, with recall lower than 0.1, due to its post-filtering strategy. VBASE performs better on large-range queries. Compared to PASE, its performance on small ranges is significantly improved, achieving higher recall despite very low QPS.

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 evaluate the robustness of the RF-ANNS algorithm in different scenarios, we add an OOD dataset to the three standard datasets and test performance under large-range queries. As shown in Figure 14, most methods perform worst on the Text2Image dataset. This is because OOD queries make the k-nearest neighbor distribution more sparse. As a result, the search needs to visit more nodes, which leads to an exponential growth in the search space and a sharp drop in efficiency.

WinFilter shows the worst performance on the WIT dataset. This suggests that its performance is sensitive to the data dimensionality. ACORN has large performance differences across datasets, indicating weak robustness.

Among vector databases, Milvus performs well. VBASE is sensitive to datasets and achieved recall below 0.8 on Text2Image. PASE is excluded from this round of tests because it does not support datasets with dimensions over 512 and performed poorly in earlier experiments.

It is worth noting that most methods achieve similar performance on the Deep and YT-Audio datasets, which have similar LID values. This shows that RF-ANNS performance is largely affected by both the LID and the dimensionality of the dataset. A high LID and high dimensionality make hybrid queries more difficult.

However, even though Deep and YT-Audio have similar LID values, the DSG algorithm shows large performance differences between them. This means that DSG is more sensitive to changes in data distribution. Its robustness is lower than that of other methods, possibly because it depends not only on LID but also on other complex data characteristics.

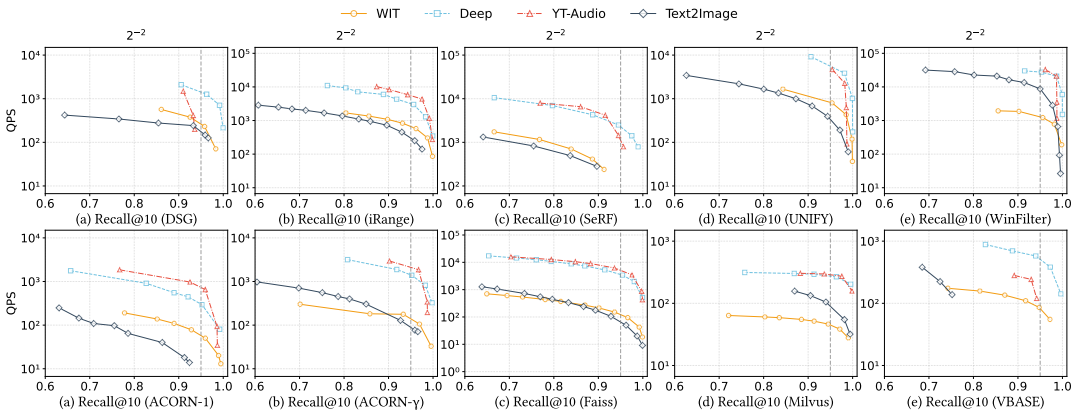


Figure 14: Dataset Effect on RF-ANN Search

Table 4: Algorithm Recommendation per Scenario

Scenario	Attribute Filtering	Range Filtering
S1: Large-scale Datasets	Puck, UNG	iRange
S2: Fast Index Construction	ACORN-1, UNG	Faiss
S3: Query on Hard Datasets	CAPS, UNG	WinFilter, iRange, DSG
S4: Query on Easy Datasets	StitchedVamana, NHQ, UNG	WinFilter, iRange
S5: General-purpose Use	Milvus, Faiss	DSG, UNIFY
S6: Resource-constrained Environments	CAPS, Faiss	Faiss, SeRF
S7: High-Recall Querying	UNG	WinFilter
S8: OOD scenarios	UNG	WinFilter

5 DISCUSSION

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

5.1 Recommendations.

5.1.1 ***Attribute Filtering.*** We summarize the applicability of different algorithms across various attribute filtering scenarios in Table 4. Puck and UNG excel on large-scale datasets (S1). ACORN-1 and UNG achieve fast index construction (S2). UNG performs well on different datasets (S3, S4). In addition, CAPS performs well on hard datasets with high LID (S3), while NHQ and StitchedVamana perform better on easy datasets with low LID (S4). Faiss and CAPS can generate relatively compact indexes with low memory usage, which is suitable for resource-constrained situations (S6). Milvus and Faiss serve as general-purpose solutions (S5). UNG stands out for its higher recall, precision, and QPS, and performs well in the OOD scenario (S7, S8).

5.1.2 ***Range Filtering.*** Table 4 also shows the application scenarios of range filtering. iRange performs well on large-scale datasets, combining efficient construction and excellent query capabilities (S1). Faiss performs well in fast index construction (S2). When processing hard datasets, WinFilter, iRange, and DSG are recommended (S3), while WinFilter and iRange also perform well on easy datasets (S4). For general applications, DSG supports dynamic updates, while UNIFY provides flexible filtering strategies (S5). Faiss and SeRF show superior memory efficiency (S6). For static queries with high recall, WinFilter is the best choice (S7). In addition, WinFilter also performs well in OOD scenarios (S8).

5.2 Challenges

5.2.1 ***Attribute Filtering.*** Most existing attribute filtering algorithms are graph-based, they typically outperform other methods in terms of query accuracy and efficiency. 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 imposes 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 filtering 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.

In our experiments, many range filtering algorithms perform worse than Faiss at high recall (0.95), showing that current algorithm designs still need improvement. We suggest future future range filtering algorithms include Faiss as a baseline.

It is also worth noting that range filtering and attribute filtering are both forms of hybrid query. 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 different algorithms, vector databases, and vector libraries. For attribute filtering algorithms, we design and conduct a series of experiments to comprehensively assess their overall performance. To facilitate fair and reproducible comparisons in future research, we enrich existing datasets with attribute values and design a variety of experimental scenarios based on practical needs. We then perform experiments on 7 real-world datasets to analyze the effectiveness of attribute filtering algorithms in depth. Additionally, we compare existing range filtering algorithms using 4 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] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. 2016. YouTube-8M: A Large-Scale Video Classification Benchmark. *CoRR* abs/1609.08675 (2016).

[3] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2015. Estimating Local Intrinsic Dimensionality. In *KDD*. ACM, 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. *J. ACM* 45, 6 (1998), 891–923.

[8] Martin Aumüller and Matteo Ceccarello. 2019. Benchmarking Nearest Neighbor Search: Influence of Local Intrinsic Dimensionality and Result Diversity in Real-World Datasets. In *EDML@SDM (CEUR Workshop Proceedings)*, Vol. 2436. CEUR-WS.org, 14–23.

[9] Artem Babenko and Victor Lempitsky. 2023. Benchmarks for Billion-Scale Similarity Search.

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

[11] Jeffrey S. Beis and David G. Lowe. 1997. Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In *CVPR*. IEEE Computer Society, 1000–1006.

[12] Yuzheng Cai, Jiayang Shi, Yizhuo Chen, and Weiguo Zheng. 2024. Navigating Labels and Vectors: A Unified Approach to Filtered Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 6 (2024), 246:1–246:27.

[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. In *NeurIPS*. 5199–5212.

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

[16] Artem Babenko Dmitry Baranchuk. 2021. Text-to-Image dataset for billion-scale similarity search. Retrieved August 23, 2023 from https://research.yandex.com/datasets/text-to-image-dataset-for-billion-scale-similarity-search

[17] Wei Dong, Moses Charikar, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*. ACM, 577–586.

[18] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *CoRR* abs/2401.08281 (2024).

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

[20] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB*. Morgan Kaufmann, 518–529.

[21] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In *WWW*. ACM, 3406–3416.

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

[23] Ben Harwood and Tom Drummond. 2016. FANNG: Fast Approximate Nearest Neighbour Graphs. In *CVPR*. IEEE Computer Society, 5713–5722.

[24] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *STOC*. ACM, 604–613.

[25] Masajiro Iwasaki. 2016. Pruned Bi-directed K-nearest Neighbor Graph for Proximity Search. In *SISAP (Lecture Notes in Computer Science)*, Vol. 9939. 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] 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.

[28] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547.

[29] 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 *Middleware Industry*. ACM, 9–16.

[30] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Trans. Knowl. Data Eng.* 32, 8 (2020), 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. *CoRR* abs/2412.02448 (2024).

[32] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, and Youlong Cheng. 2022. Monolith: Real Time Recommendation System With Collisionless Embedding Table. *CoRR* abs/2209.07663 (2022).

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

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

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

[36] 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. *Proc. ACM Manag. Data* 1, 2 (2023), 197:1–197:25.

[37] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proc. ACM Manag. Data* 2, 3 (2024), 120.

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

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

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

[41] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, and et al. 2024. Results of the Big ANN: NeurIPS'23 competition. *CoRR* abs/2409.17424 (2024).

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

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

[44] 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.* 46, 3 (2023), 39–54.

[45] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-NN graph construction for visual descriptors. In *CVPR*. IEEE Computer Society, 1106–1113.

[46] Jianguo Wang, Xiaomeng Yi, Rentong Guo, and et al. 2021. Milvus: A Purpose-Built Vector Data Management System. In *SIGMOD Conference*. ACM, 2614–2627.

[47] 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. In *NeurIPS*.

[48] 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. In *NeurIPS*.

[49] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 14, 11 (2021), 1964–1978.

[50] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*. Morgan Kaufmann, 194–205.

[51] 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. *Proc. VLDB Endow.* 13, 12 (2020), 3152–3165.

[52] Yuexuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S. Jensen. 2024. iRangeGraph: Improvising Range-dedicated Graphs for Range-filtering Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 6 (2024), 239:1–239:26.

[53] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension. In *SIGMOD Conference*. ACM, 2241–2253.

[54] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. 2023. VBASE: Unifying Online Vector Similarity Search and Relational Queries

via Relaxed Monotonicity. In *OSDI*. USENIX Association, 377–395.

[55] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 1 (2024), 69:1–69:26.