An Experimental Evaluation of Hybrid Querying on Vectors (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 either fail to cover mainstream algorithms or omit systematic comparisons or in-depth analysis on different methods. To address this issue, we design a comprehensive 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, we construct standardized attribute and range sets for attribute filtering and range filtering, respectively, enabling a unified comparison of algorithms in terms of index construction efficiency and query performance. Furthermore, we evaluate the robustness of the algorithms across multiple dimensions, including data distributions, platforms, and scalability on a 100-million-scale dataset. Additionally, we conduct an in-depth analysis of the experimental results based on the underlying principles of algorithms. Extensive experimental results reveal the strengths and weaknesses of each algorithm. Based on the 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 Querying on Vectors (EA&B). PVLDB, 19(2): XXX-XXX, 2025.
doi:XX.XX/XXX.XX
PVLDB Artifact Availability:

Proceedings of the VLDB Endowment, Vol. 19, No. 2 ISSN 2150-8097.
doi:XX.XX/XXX.XX

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

1 INTRODUCTION

Nearest Neighbor (NN) search [14] 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 [30] and image retrieval [42]. However, with the exponential growth of data volume and the increasing dimensionality of vectors [46], traditional NN search methods struggle to meet the demands of real-time search. To address this issue, studies turn to more efficient approaches - Approximate NN (ANN) search [7, 12, 20, 32], which 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) [41]. When the constraint involves a specific attribute value, this problem refers to Attribute Filtering Approximate Nearest Neighbor (AF-ANN) search [21, 44]. 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 [48, 51]. For instance, filtering clothes priced between 100 and 200. To meet these application demands, hybrid querying techniques [27, 47] 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 algorithms [40, 44] and range filtering algorithms [48, 51]. In practical applications, the performance of hybrid query algorithms is influenced not only by unstructured

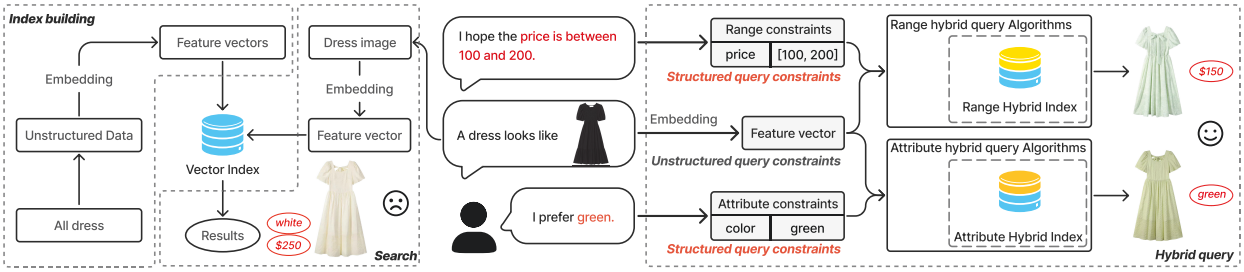


Figure 1: Hybrid Query Example

data but also closely relates to structured data. For attribute filtering algorithms, factors such as the number of attributes, attribute distribution [13], and attribute selectivity [43, 47] significantly impact algorithm performance. As for range filtering algorithms, the width of the query range and the characteristics of different datasets play an important role in determining algorithm performance.

Although existing studies [9, 45] conduct benchmark evaluations of ANN methods, a systematic experimental study specifically targeting hybrid query algorithms is still lacking. Despite BigANN 2023 [38] evaluates the performance of some hybrid query algorithms, it still presents the following notable limitations: 1) The evaluation covers a limited number of algorithms and fails to comprehensively include mainstream methods. 2) It only uses a single dataset and focuses solely on single-attribute and dual-attribute query scenarios, while not considering the impact of attribute selectivity on query performance. 3) It only simply compares the query speed and recall of different algorithms, but does not consider key metrics such as the time and space overhead of each algorithm, and it lacks an in-depth analysis of the specific reasons for algorithm performance.

To fill this gap, we conduct the first comprehensive experimental evaluation of hybrid query algorithms, systematically analyzing their index construction costs and query efficiency across diverse scenarios. Furthermore, we rigorously assess the robustness of each algorithm under different experimental settings, thereby providing valuable insights for future research and practical applications.

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 5 aspects.

(1) **Systematic Classification and Overview.** We systematically classify 6 representative attribute filtering algorithms along multiple dimensions, including index organization, filtering strategies, Boolean logic support, and index construction methods. In addition, we survey 5 range filtering algorithms, and provide an overview of one vector library and 3 vector databases. These efforts offer a unified reference framework for future research.

(2) **Strengthening 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. We design comprehensive experimental settings that reflect diverse application requirements, including varying vector datasets, attribute distributions, number of attributes participating in index construction, number of query attributes, and selectivity. Furthermore, to further investigate the scalability and adaptability of the algorithms, we also conduct experiments on a large-scale dataset and two multimodal datasets. 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.

(3) **Evaluation of Attribute Filtering Algorithms.** We systematically evaluate 6 attribute filtering algorithms, 1 library, and 3 databases on 11 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 selectivity 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 algorithms that support range filtering on 6 real-world datasets, using varying query range settings in the experiments. The experimental results show the performance of these algorithms in terms of index construction efficiency, storage overhead, and query performance. We also further analyze the factors that affect the performance of the range filtering algorithms.

(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 query. These challenges include limited Boolean logic support, the lack of multi-attribute range filtering capabilities, and the sensitivity of algorithms to selectivity. 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 two categories: 1) Attribute Filtering Nearest Neighbor (*AF-NN*) search. 2) Range Filtering Nearest Neighbor (*RF-NN*) search.

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 v_i 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 $D = \{(v_1, a_1), \ldots, (v_n, a_n)\}$ be a dataset of n d-dimensional vectors, each v_i 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 Query

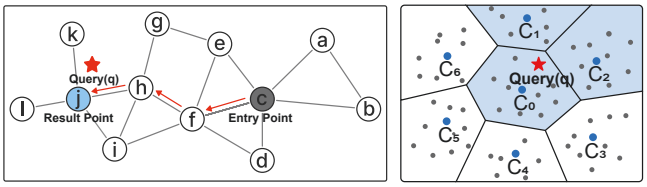
Current hybrid query methods mainly adopt graph-based [16, 19, 23, 25, 31] or Inverted File Index (IVF)-based [26] index organization. Here, we briefly review the core concepts of these indexes and outline how attribute and range constraints are embedded to support hybrid queries functionality.

Graph. Graph structures accelerate ANN search by connecting each data point with its nearest neighbors, as shown in Figure 2a. During querying, algorithms start from an initial point and use a greedy strategy to progressively move towards closer neighbors until convergence (e.g., the path from c to j in the 1NN case).

Graph indexes are widely used in hybrid ANN search due to their efficiency. To support hybrid queries, in attribute filtering, a common practice involves attaching attribute metadata to nodes or edges, thereby skipping parts that do not satisfy query conditions during traversal (e.g., NHQ, Filtered-DiskANN, ACORN). In range filtering, edges typically store valid attribute intervals, and traversal proceeds only along edges that satisfy range constraints, thus narrowing the search space (e.g., SeRF, DSG).

IVF. IVF divides data into multiple clusters through a clustering algorithm (e.g., K-Means), with each cluster represented by a center (Figure 2b). During querying, algorithms search only a few clusters closest to the query point, which reduces computational load.

IVF-based indexes also offer unique advantages in hybrid ANN search scenarios, including natural support for pre-filtering, low memory footprint, and small index size. Efficient filtering is achieved by excluding vectors that do not satisfy attribute constraints before distance computation (e.g., Faiss, Milvus). Furthermore, hierarchical partitioning by attributes can form finer-grained sub-clusters to enhance filtering efficiency (e.g., CAPS, Puck).



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

Figure 2: Graph Index and IVF Index

3 HYBRID QUERYING ALGORITHMS

Table 1 provides a comparative overview of 6 attribute filtering and 5 representative range filtering algorithms.

3.1 Attribute Filtering Algorithms

NHQ [1] **[44]** 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 [2] **[21]** Filtered-DiskANN also supports simultaneous filtering. Built upon the Vamana [40] 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. Filtered-DiskANN proposes two indexes: 1) **FilteredVamana**, which incrementally builds the graph index by inserting data points and dynamically adding edges, allowing adaptive expansion. 2) **Stitched-Vamana**, which adopts a batch construction strategy—constructing separate Vamana subgraphs for each attribute, followed by merging and edge pruning.

CAPS [3] **[22]** 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. It 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 [4] **[35]** ACORN adopts a predicate-agnostic indexing framework. It is based on Hierarchical Navigable Small World (HNSW)

[1] https://github.com/KGLab-HDU/TKDE-under-review-Native-Hybrid-Queries-via-ANNS
[2] https://github.com/microsoft/DiskANN
[3] https://github.com/gaurav16gupta/constrainedANN
[4] https://github.com/stanford-futuredata/ACORN

Table 1: Overview of AF-ANN and RF-ANN Search Algorithms

Characteristics/Algorithms		NHQ	Filtered	Stitched	CAPS	ACORN	UNG	Puck	DSG	iRange	SeRF	UNIFY	Win
AF	AND	Y	N	N	Y	Y	Y	Y	AF Not Support				
	OR	N	Y	Y	N	Y	Y	Y					
	Flexible Attributes	N	Y	Y	Y	Y	Y	Y					
	Complex Boolean	N	N	N	N	Y	N	N					
RF		N	N	N	N	Y	N	N	Y	Y	Y	Y	Y
Filter Type		C	C	C	C	B	B	B	B	B	B	A/B/C	A
Dynamic Insert		N	Y	N	Y	Y	Y	Y	N	Y	Y	Y	Y
Multi Thread		N	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y
Disk Support		N	Y	N	N	N	N	N	N	N	N	N	N
Base		Graph	Graph	Graph	IVF	Graph	Graph	IVF	Graph	Graph	Graph	Graph	Graph

Filtered: FilteredVamana, *Stitched*: StitchedVamana, *Win*: WinFilter. *AND* / *OR*: Whether the algorithm supports the corresponding logical operations. *Flexible Attributes*: Allows the number of query attributes to differ from that in index construction. *Complex Boolean*: Supports NOT or nested Boolean expressions. *Dynamic Insert*: Supports dynamic data insertion. *Multi Thread*: Supports multi-threaded search. *Disk Support*: Supports disk-based storage. *Base*: Underlying ANN index structure used. *A*: Post-filtering (Filter after ANN search), *B*: Pre-filtering (Filter before ANN search), *C*: Simultaneous filtering (Filter simultaneously with ANN search), *Y*: Support, *N*: Unsupport.

[33] and builds a denser hierarchical graph through neighborhood expansion and edge pruning. During search, it filters out nodes violating constraints, maintaining a valid nearest neighbor graph. 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 [5] **[13]** UNG serves as a unified framework that integrates various graph-based ANN indexes to support hybrid queries. 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.

Puck [6] **[11]** 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

SeRF [7] **[51]** 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 [8] **[18]** 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 [9] **[48]** 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 [10] **[36]** 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. The system considers only edges valid for current attribute range to boost efficiency during search.

UNIFY [11] **[29]** 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. 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.

3.3 Vector Libraries and Databases

The vector library and databases discussed in the following are not explicitly designed for hybrid query, but they support both attribute filtering and range filtering.

Faiss[12] **[17]** Faiss is an efficient similarity search library. It supports hybrid search by using an ID selector to filter data before performing ANN search. In Faiss, we use the HQI batch optimization strategy in the IVF index. It groups queries with the same filter conditions. Each group only runs the filter once. It uses efficient matrix operations to compute vector similarity in batches.

[5] https://github.com/YZ-Cai/Unified-Navigating-Graph
[6] https://github.com/baidu/puck
[7] https://github.com/rutgers-db/SeRF
[8] https://github.com/JoshEngels/RangeFilteredANN
[9] https://github.com/YuexuanXu7/iRangeGraph
[10] https://github.com/rutgers-db/DynamicSegmentGraph
[11] https://github.com/sjtu-dbgroup/UNIFY
[12] https://github.com/facebookresearch/faiss

PASE [13] **[49]** PASE is a vector index plugin for PostgreSQL. It uses a post-filtering strategy. Since the candidate set size is fixed, it may fail to return enough top-k results.

VBASE [14] **[50]** VBASE is also a vector plugin for PostgreSQL. Unlike PASE, it applies filters during the HNSW index traversal, discarding unmatched nodes on the fly. This strategy ensures that enough valid results are returned .

Milvus [15] **[43]** Milvus is a vector database designed for large-scale similarity search. In hybrid queries, it adopts a pre-filtering strategy, similar to the ID selector mechanism used in Faiss.

4 EXPERIMENTS

4.1 Experimental Setup

4.1.1 Datasets. Our experiments employ 12 real-world datasets covering diverse domains: Audio [6], Enron [39], GIST1M [4], GloVe [37], Msong [5], SIFT1M [4], Deep1M [10], WIT-Real [1], YT-Audio-Real [2], Text2image [15], Text2image-Mix, and Deep100M.

Table 2 summarizes the key characteristics of all datasets and their applicable filtering scenarios. In particular, we report the Local Intrinsic Dimensionality (LID) [28], which is a commonly used metric to quantify dataset hardness. Following the standard evaluation protocol [3], we randomly sample 10,000 data points from each dataset to compute LID. A higher LID value indicates a more complex dataset.

The first 7 datasets are widely used in existing studies, while the remaining are common datasets with added real attributes. As they do not inherently come with attributes, we generate attributes for them to precisely control attribute quantity and distribution. WIT-Real and YT-Audio-Real datasets use real attributes. For the WIT-Real dataset, we use the CLIP model to encode the page title into vectors, and extract the language type and the length of the context page description as real attributes. The YT-Audio-Real dataset uses YouTube8M data, employing the video category ID and view counts as real attributes.

Additionally, we use the Out-of-Distribution (OOD) dataset Text2-Image and the large-scale dataset Deep100M to evaluate the generalization capability of the algorithms. The Text2Image dataset includes two versions. The original version contains 1 million image embeddings as the base dataset and uses 10 thousand text embeddings as the query set. The Text2Image-Mix version is composed of 0.5 million image embeddings and 0.5 million text embeddings as the base dataset, with a query set of 5 thousand image embeddings and 5 thousand text embeddings.

4.1.2 Evaluation Metrics. To assess the overall performance of different algorithms, we adopt a multi-dimensional quantitative evaluation framework [45], including 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.

[13] https://github.com/alipay/PASE
[14] https://github.com/microsoft/MSVBASE
[15] https://github.com/milvus-io/milvus

Table 2: Datasets

Dataset	Dimension	Base Data	Queries	LID	Type	Used
Audio	192	53,387	200	14	Audio	AF
Enron	1369	94,987	200	23	Text	AF
GIST1M	960	1,000,000	1,000	45	Image	AF
GloVe	100	1,183,514	10,000	47	Text	AF
Msong	420	992,272	200	23	Audio	AF
SIFT1M	128	1,000,000	10,000	19	Image	AF
Deep1M	96	1,000,000	10,000	22	Image	RF
WIT-Real	512	1,000,000	40,300	48	Image	Both
YT-Audio-Real	128	1,000,000	10,000	15	Audio	Both
Text2Image	200	10,000,000	10,000	59	Text,Image	Both
Text2Image-Mix	200	1,000,000	10,000	21	Text,Image	Both
Deep100M	96	100,000,000	10,000	22	Image	Both

(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 and search.

4.1.3 Setting. We conduct most experiments on a server running Ubuntu 20.04, equipped with an AMD EPYC 7K62 processor (2.6GHz) featuring 256GB of RAM and 192 MiB of L3 cache. We test on large-scale dataset using a server with 2×AMD 7Y43 processors and 1TB RAM. Unless otherwise stated, we execute index construction with 32 threads to accelerate the process [8]. The queries are all executed on a single thread.

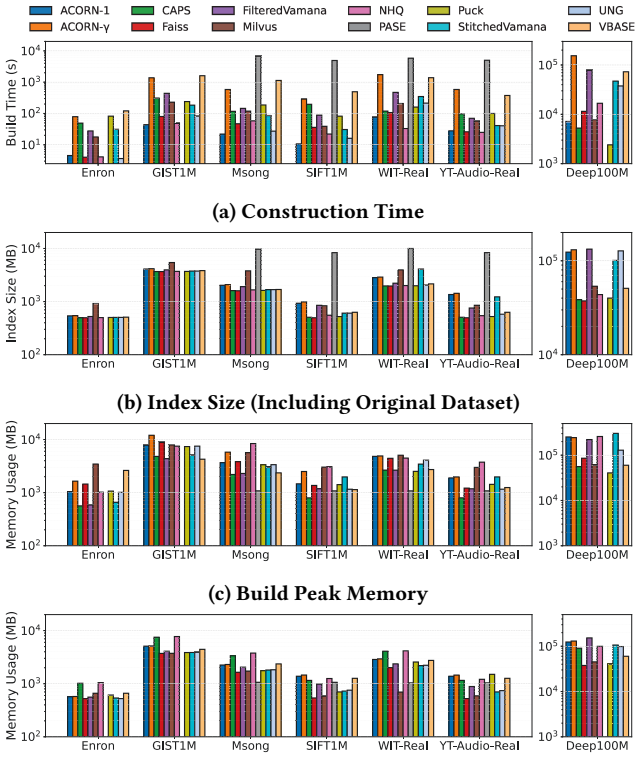
Regarding experimental parameter selection, we adopt the recommended settings from original papers when available. For datasets without suggested parameters, we test different configurations and select relatively optimal ones for final evaluation. Due to the large number of datasets and algorithms involved, specific parameters are not individually listed in the paper but are fully provided in our code repository.

4.2 Attribute Filtering

4.2.1 Time and Space overhead. To investigate the time and space overhead of different algorithms, we evaluate 4 key metrics in the single-attribute scenario: index construction time, index size, build peak memory and search peak memory. Notably, PASE on GIST1M and Enron is omitted, as it supports only data with dimensionality less than 512. Furthermore, we did not measure PASE on large-scale datasets, as we estimated it would take over 10 days. We use 64 threads to build the index for the large-scale Deep100M dataset, and 32 threads for the other 6 medium-scale datasets. Notably, the VBASE and PASE algorithms only support single-threaded construction.

Index construction time. On the medium-scale datasets (Figure 3a left), among the algorithms limited to single-threaded construction, PASE has the longest build time. This is because its on-demand loading and caching strategies frequently trigger disk I/O and page scheduling, which seriously affects efficiency. In contrast, VBASE uses full-memory builds and is therefore comparatively faster. Among the multi-threaded algorithms, ACORN-γ is the slowest because it needs to expand the neighbor list and evaluate a large number of candidates. Its build time on some datasets even exceeds that of the single-threaded VBASE.

On the large-scale dataset Deep100M (Figure 3a right), in addition to the three methods already mentioned, we also observe



(a) Construction Time

(b) Index Size (Including Original Dataset)

(c) Build Peak Memory

(d) Search Peak Memory

Figure 3: Time and Space Overhead (AF)

that algorithms based on Vamana (FilteredDiskANN, StitchedVamana and UNG) exhibit significantly increased construction time. This is because Vamana adopts a two-pass graph construction and optimization strategy (greedy search + robust pruning), resulting in extremely high computational overhead. In contrast, the construction time of IVF-based algorithms grows much more slowly with dataset size. This is because their shard-based design and high parallel efficiency, making them more scalable and advantageous in large-scale scenarios.

Index size. As shown in Figure 3b, PASE exhibits significantly larger index sizes across all supported datasets, mainly due to its reliance on a large amount of metadata. For medium-scale datasets, other methods show moderate variation. However, on large-scale dataset, Graph-based algorithms such as ACORN, FilteredDiskANN, and UNG tend to exhibit a sharp increase in index size, primarily due to the large number of candidate neighbors they maintain. Overall, IVF-based methods tend to produce more compact indexes.

Peak memory. We present the peak memory usage during index construction and query processing in Figure 3c and Figure 3d, respectively. We observe that peak memory usage is generally correlated with the combined size of the index and the dataset. Methods that produce larger indexes tend to consume more memory during both construction and querying, and this trend holds consistently across different datasets.

However, there are notable exceptions. During index construction, PASE adopts an on-demand loading strategy, resulting in build peak memory usage even lower than the final index size. NHQ generates a large number of candidate neighbors during construction but retains only a small subset per node in the final index, leading to high peak memory usage despite a relatively compact index. During querying, CAPS maintains a large candidate set dynamically, which significantly increases its search peak memory consumption.

4.2.2 Performance Evaluation. We next evaluate the query performance of AF-ANN search methods, focusing on 3 main scenarios.

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

As shown in Figure 4, as recall approaches 1, search often requires traversing larger candidate sets or deeper graph paths. This leads to increased computational cost and a sharp QPS drop for most methods. Despite this, UNG consistently maintains high QPS due to its LNG structure, which eliminates unnecessary online filtering of irrelevant attributes and searches only within the subgraph formed by points satisfying the filtering conditions, thereby improving query efficiency.

NHQ and StitchedVamana also perform well and show similar and stable performance across all datasets. This indicates that joint filtering and search strategies can achieve consistent effectiveness under high-recall requirements. StitchedVamana outperforms FilteredVamana in query efficiency. This difference stems 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. Additionally, Puck and CAPS also perform relatively well. They further reduce the search space on IVF indexes by using attribute partitioning, thereby achieving solid search performance.

Among all methods, vector database systems (PASE, VBASE, Milvus) show the worst performance. 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-Attribute Index Construction 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.

Faiss offers an ID filtering mechanism, it operates solely during query execution and does not influence index building. ACORN is designed for single-attribute indexing. In database systems (VBASE, PASE, Milvus), attributes do not participate in index construction. Hence, they are excluded from this comparison.

(1) *Effect of Multi-Attribute Building on Algorithm Performance (M vs. S).* As shown in Figure 5, Puck demonstrates nearly no performance degradation. CAPS shows a slight performance decline due to the increased query overhead caused by excessive attribute-based grouping. In contrast, 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

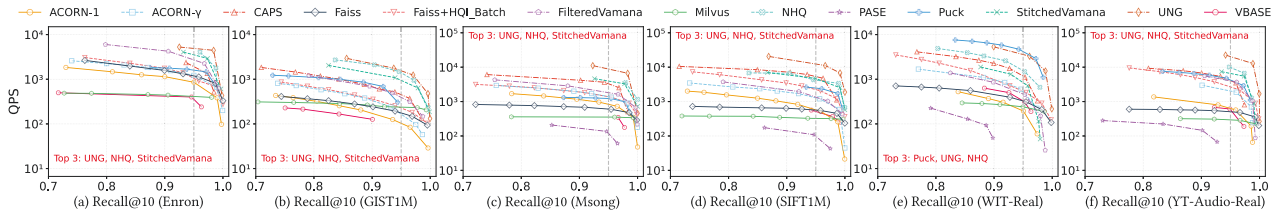


Figure 4: Performance of Single-Attribute Index Construction and Single-Attribute Query

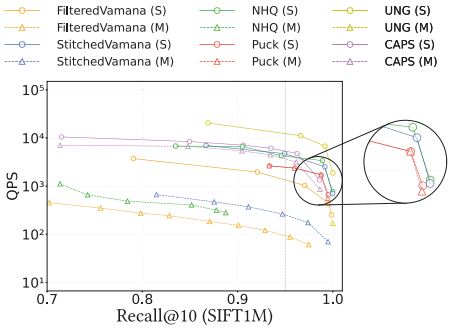


Figure 5: Effect of Multi-Attribute Index Construction on Single-Attribute Query Performance (*S* denotes single-attribute index construction and single-attribute query, *M* denotes multi-attribute index construction and single-attribute query)

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.

(2) *Performance of Algorithms under Multi-Attribute Building (M vs. M).* As shown in Figure 5, IVF-based methods (Puck and CAPS) significantly outperform graph-based methods. The reason for their superior performance is that they filter out irrelevant results during the retrieval process rather than searching for more candidates. In contrast, multi-attribute construction significantly degrades the performance of graph-based methods, for which we identify the primary reason. For algorithms with tightly coupled attribute and graph structures, increasing the number of attributes during index construction increases graph complexity. This makes single-attribute queries more difficult and degrades performance.

Multi-Attribute Index Construction and Multi-Attribute Query. Compared to single-attribute filtering, multi-attribute joint filtering better reflects real-world scenarios. To evaluate algorithm performance under this setting, we construct the index using 3 uniformly distributed attributes and apply filtering on all 3 attributes simultaneously during queries. In this experiment, we only evaluate algorithms that support this multi-attribute query scenario.

As shown in Figure 6, UNG still maintains the best performance across all datasets. Since some strong competing algorithms do not support multi-attribute querying, CAPS becomes the second-best algorithm, but its performance on small datasets is relatively poor. This is because CAPS performs multi-level partitioning based on attributes on top of the IVF index, which introduces additional overhead. While this overhead negatively impacts performance on small datasets, on large datasets, the overhead is amortized, and the increasing dataset scale makes the advantages of attribute-aware partitioning evident.

NHQ performs moderately overall in multi-attribute joint filtering scenarios but poorly on small datasets. Its reliance on fusion distance computations makes it highly sensitive to the number of attributes, impacting both efficiency and accuracy under multi-attribute filtering. For IVF-based methods not specifically optimized for hybrid queries (Faiss and Milvus), their QPS shows a similar and relatively stable trend as recall varies. Among them, Milvus, as a general-purpose system, exhibits comparatively poor performance.

4.2.3 Robustness. In the following, we evaluate the robustness of these algorithms.

Attribute Distribution. To evaluate the impact of attribute distribution, we generate base and query attributes on the SIFT dataset using four representative distributions: long-tail, normal, power-law, and uniform. As illustrated in Figure 7, the evaluated algorithms exhibit varying degrees of sensitivity to attribute distribution shifts. PASE is excluded due to poor past performance. The proximity of the curves in the figure reflects algorithm sensitivity to attribute distributional variations. When the curves converge closely, it demonstrates that the algorithm maintains stable performance when attribute distributions change.

textcolorblack Algorithms that are highly sensitive to attribute distribution tend to rely passively on the statistical characteristics of the attributes in their design. For instance, graph-based indexing methods like FilteredVamana and NHQ are prone to degraded connectivity or distorted structures under power-law distributions, where label frequencies are highly imbalanced. Similarly, VBASE, which employs a post-filtering strategy, shows efficiency that directly correlates with attribute selectivity—the more selective the filter, the more computational resources are wasted on retrieving irrelevant vectors. Faiss, which searches within the nearest clusters, may need to traverse more clusters to find sufficient results when relevant data points are unevenly distributed across clusters due to attribute skew, which impacts performance.

In contrast, algorithms with low sensitivity to attribute distribution incorporate smarter indexing designs or adaptive query strategies to mitigate these challenges. Milvus, for example, leverages a built-in query optimizer that dynamically determines whether to filter by attributes or search vectors first, based on the selectivity of the query, thus ensuring more robust performance. Other approaches enhance robustness by separating attribute and vector indexing into distinct stages: UNG uses a logical graph to navigate

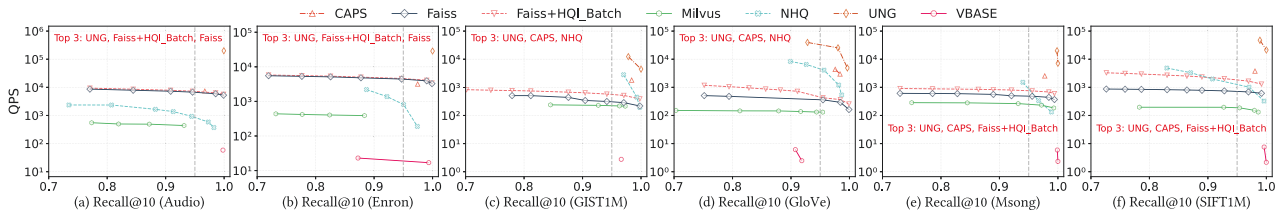


Figure 6: Performance of Multi-Attribute Index Construction and Multi-Attribute Query

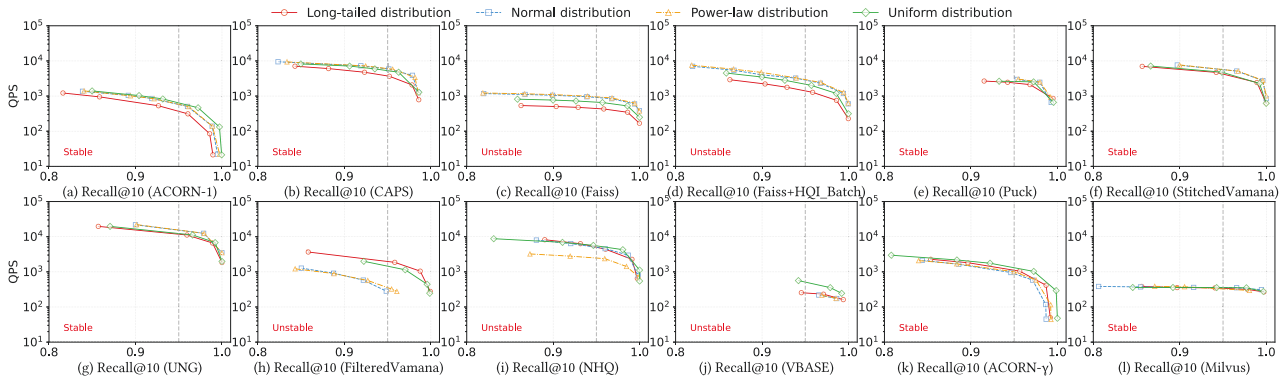


Figure 7: Effect of Attribute Distribution on Query Performance

attribute relationships, while CAPS employs an attribute frequency tree to adapt to power-law patterns. For ACORN–γ, during index construction, attributes are primarily used for pruning in the lowest-layer graph, while the upper-layer hierarchical navigation graphs remain largely unaffected. Consequently, changes in attribute distribution have minimal impact on overall performance.

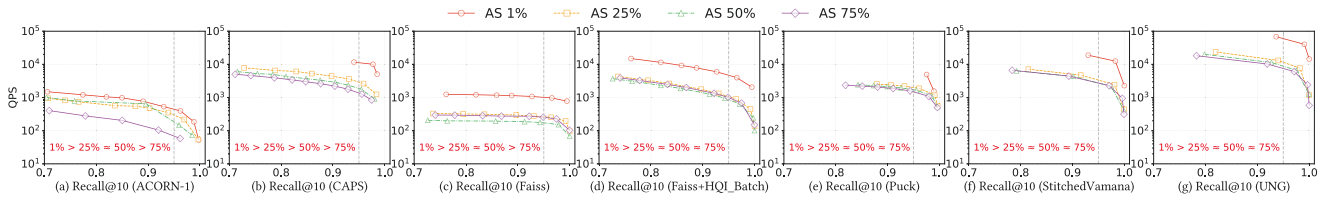
Single-Attribute Selectivity. In hybrid queries, varying attribute selectivity (AS) can significantly impact computational cost and query efficiency. AS is defined as the proportion of data points sharing a given attribute value. For example, AS of 1% indicates that only 1% of the dataset meets the given attribute. To investigate this effect, we evaluate 4 AS settings (1%, 25%, 50%, and 75%) on the SIFT1M dataset. Based on the query performance trends of different algorithms under varying AS, we categorize the algorithms into 3 groups:

(1) *Algorithms optimized for low AS.* Figure 8a shows the algorithms suitable for low AS. ACORN-1, Faiss, Puck, and UNG all use pre-filtering strategies to shrink the search space in low AS, leading to better performance. Pre-filtering strategies significantly reduce the number of candidate points under low AS, thereby lowering the cost of computations. As a result, such strategies tend to perform better in low AS scenarios. 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, it requires broader global exploration.

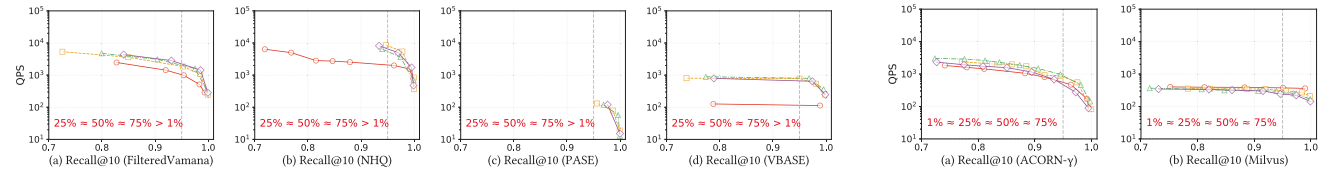
(2) *Algorithms optimized for high AS.* Figure 8b shows the algorithms suitable for high AS. At low AS, FilteredVamana performs poorly in these scenarios. Its dynamic pruning strategy makes it difficult to balance vector distance and attribute relevance, resulting in overly restricted search paths. NHQ struggles to locate relevant regions efficiently, leading to excessive computations on non-matching nodes. 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. VBASE relies on post-filtering, which increases computational overhead at low AS.

(3) *Algorithms insensitive to AS.* Figure 8c shows algorithms that are insensitive to AS. 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 settings 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.

Multi-Modal Queries. We conduct experiments on two multi-modal datasets Text2image-Mix and Text2image. On Text2image-Mix dataset, as shown in Figure 9, UNG and NHQ consistently perform best, followed by CAPS. Although the dataset contains two modalities, due to the significant differences between the modalities, they are actually two separate regions in the vector space. The performance is similar to that of the single-modal dataset in Figure 4. This is because mixed multi-modal queries are actually the same as conducting experiments on two datasets of the same modality. This is not actually a difficult query. In contrast, the Text2image dataset shown in Figure 9 is different. The query set and the base dataset belong to two completely different modalities. Queries are far from the dataset points, and the ground truths are more dispersed. To find the ground truths, algorithms typically need to examine more points, which leads to degraded performance.



(a) Algorithms Optimized for Low AS



(b) Algorithms Optimized for High AS **(c) Algorithms Insensitive to AS**

Figure 8: Performance of Different Attribute Selectivity (AS) under the Same Algorithm

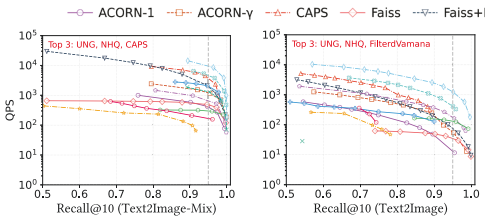


Figure 9: Multi-Modal Datasets (AF)

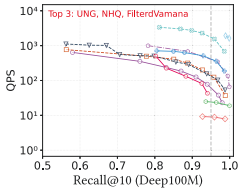


Figure 10: Large-Scale Dataset (AF)

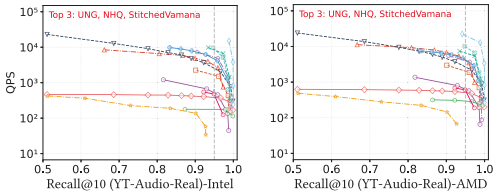


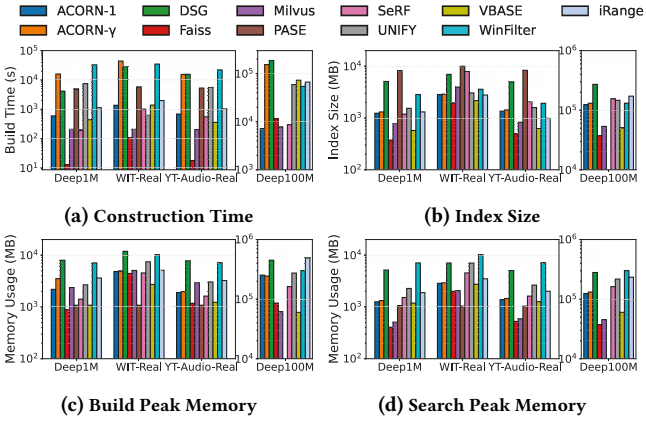
Figure 11: Different Platforms (AF)

Large-scale Dataset. As shown in the Figure 10, on the Deep100M dataset, UNG and NHQ still exhibit excellent performance, demonstrating good adaptability to large-scale datasets. However, this also comes with significant memory overhead. The clustering-based algorithm Faiss+HQI_Batch shows moderate performance but has very low memory overhead, which also has certain practical significance. Furthermore, we do not present the results of StitchedVamana and CAPS, as their recall remained consistently low. StitchedVamana sets the search entry point to 0 by default, which makes it prone to getting stuck in local optima when the entry is far from the target region, hindering effective navigation. CAPS, on the other hand, cannot properly run large-scale dataset experiments because its code implementation does not account for such scenarios.

Different Hardware Environments. To investigate the impact of different server configurations on the performance of the algorithms, we conduct additional experiments on an Ubuntu 20.04 server equipped with dual Intel(R) Xeon(R) Gold 6330 CPUs @ 2.00GHz featuring 128GB of RAM and 84 MiB of L3 cache using the YT-Audio-Real dataset, and the results are shown in Figure 11. It is not difficult to see that the relative ranking of the algorithms under the two servers essentially remains unchanged, indicating that the performance of these methods is not significantly affected by differences in CPU configurations. Due to the lower CPU frequency, reduced RAM capacity, and smaller L3 cache, all methods demonstrate slightly lower QPS values.

4.3 Range Filtering

Due to the limitations of existing range filtering algorithms, our experiment uses only a single attribute, with the dataset sorted in ascending order by value. The WIT-Real and YT-Audio-Real datasets use real attribute values, while the rest use randomly generated ones.



(a) Construction Time **(b) Index Size**

(c) Build Peak Memory **(d) Search Peak Memory**

Figure 12: Time and Space Overhead (RF)

4.3.1 Time and Space overhead. Similarly, we also evaluate 4 key metrics mentioned above in *Range Filtering* algorithms. Notably, all algorithms use single-threaded index construction on medium-scale datasets, and 64-thread construction on the large-scale Deep100M dataset, except DSG, which only supports single-threading. For WinFilter, we use its integrated Super-Postfiltering on medium datasets for best performance, and switch to its base algorithm Vamana-WST on Deep100M due to construction overhead.

Index construction time.

As shown in Figure 12a, Faiss and Milvus, based on the IVF architecture, require less time. This indexing structure enables fast construction through subset sampling and centroid computation, thereby reducing computational cost. Milvus performs particularly well on large-scale datasets, mainly due to its cache-aware optimizations, which significantly reduce CPU cache misses.

In addition, the two HNSW-based methods, ACORN-1 and SeRF, also achieve short index construction time. ACORN-1 closely follows the original HNSW construction process and reduces the number of candidate neighbors in the upper layers to lower computational overhead. SeRF attaches timestamp information to edges during construction, making the total build time comparable to that of a single HNSW graph.

We observe that ACORN-γ, DSG and WinFilter exhibit relatively long index construction time. Although the first two methods are also based on HNSW, ACORN-γ extends the neighbor list of each node to provide more candidate paths, which increases both computational and memory overhead. DSG, on the other hand, incorporates a rectangle tree structure for index management, making the construction process more complex and costly. WinFilter also incurs high construction cost, as it relies on a tree structure and builds a separate nearest neighbor graph for each node, which leads to redundant computations. However, on large-scale datasets, its preprocessing stage is replaced by the base algorithm Vamana-WST, effectively avoiding significant time overhead.

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.

Similar to attribute filtering, PASE requires the most storage. Additionally, DSG also results in a large index size, especially on large-scale datasets. As an enhancement over SeRF, DSG stores additional metadata for each edge in its index structure. While this design improves retrieval accuracy, it significantly increases the index size on massive datasets.

Peak memory. As explained in the attribute filtering analysis, methods with larger index sizes (including the dataset) tend to exhibit higher peak memory consumption. As shown in Figure 12c and Figure 12d, range filtering also adheres to this pattern.

WinFilter also presents an exception: its peak memory usage is also affected by the level of parallelism. Although the final index size remains relatively small, the peak memory usage increases significantly during both construction and search. During construction, this is caused by the parallel building of local indices for each tree node, which results in substantial temporary memory usage. Similarly, during search, WinFilter needs to load multiple local indices in parallel for different tree nodes, further contributing to high peak memory consumption.

4.3.2 Performance Evaluation. In this experiment, we follow the query range definition adopted by prior work [34]. Specifically, for a given query, the range ratio is defined as 2^{-i} when the query range covers $\frac{n}{2^i}$ data points, where n is the total dataset size.

Based on this definition, we evaluate algorithm performance under four query range settings: 2^{-2}, 2^{-4}, 2^{-6}, and 2^{-8}. For space considerations, Figure 13 reports results for only the largest (2^{-2}) and smallest (2^{-8}) range ratios.

As shown in Figure 13, the algorithms that performed well are WinFilter, iRange, UNIFY, and Faiss. WinFilter maintains optimal performance by building an index for each tree node. It then performs an ANN search on the relevant nodes and post-filters the results, ensuring high recall and QPS. In contrast, iRange dynamically constructs a search graph during querying. This method has low overhead and performs well in small-range queries, but its performance degrades as the query range increases. UNIFY utilizes a skip list for localization followed by a linear scan for searching in small-range query scenarios. Its performance is generally lower than graph-based structures like iRange in these cases. Interestingly, Faiss, aided by its efficient ID selector for filtering, calculates distances for only a small subset of points in small-range scenarios. Consequently, it can outperform algorithms specifically designed for range searches.

SeRF, ACORN, and vector databases performed poorly. SeRF demonstrates relatively weak overall performance, with a pronounced drop in recall under small-range query scenarios. This is because SeRF compresses the entire HNSW graph by attaching range information to its edges. However, during small-range queries, its multi-filtering mechanism requires scanning a large number of edge entries that are irrelevant to the current query range, introducing fixed computational and memory access overhead. Additionally, graph traversal may lead to the exploration of numerous invalid neighbors, increasing the likelihood of the search becoming trapped in local optima and further degrading performance. The overall performance of ACORN is poor. Unlike other algorithms specifically designed for range filtering, ACORN does not design an index specifically for range queries. Instead, it adopts a pre-filtering strategy, resulting in poorer performance. The performance of vector databases is also poor in range queries, as they are not specifically optimized for such queries. PASE performs particularly poorly in small range queries, with a recall rate lower than 0.1, mainly due to its post-filtering strategy.

4.3.3 Robustness. ***Multi-Modal Queries.*** Similar to attribute filtering, the performance of range filtering queries on the Text2image-Mix dataset (Figure 14) is similar to the single-modality query performance (Figure 13). However, on the Text2Image dataset, these algorithms exhibit a significant performance degradation (Figure 14). This is because most of these range query algorithms are graph-based, and such algorithms require more iterations to find true neighbors in OOD datasets.

Large-scale Dataset. As shown in the Figure 15, On the Deep100M dataset, the top 3 ranked algorithms (WinFilter, UNIFY, and iRange) remain unchanged. However, their high performance is achieved at the cost of extremely high memory consumption (Figure 12d). In contrast, VBASE and Faiss demonstrate moderate performance while consuming significantly less memory.

Different Hardware Environments. Consistent with Section 4.2.3, we conduct range filtering experiments on the YT-Audio-Real dataset using two server configurations, with results presented in Figure 16. Although the third-ranked algorithm differs, we consider the ranking order remains stable since the performance of the third and fourth algorithms is comparable. Therefore, different server configurations only affect the performance of individual algorithms, with minimal impact on their relative performance ranking.

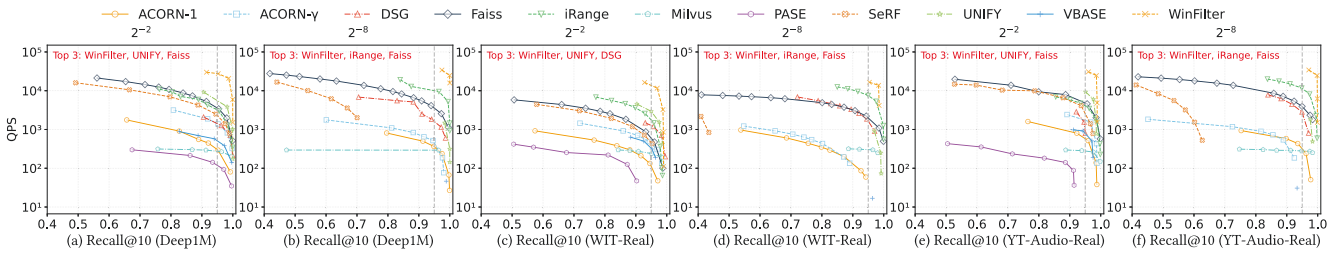


Figure 13: Performance of RF-ANN Search

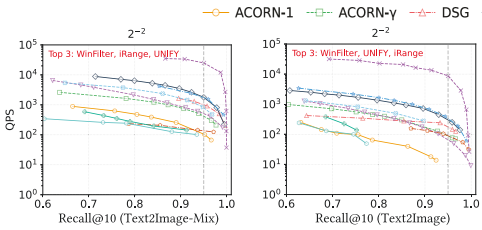


Figure 14: Multi-Modal Datasets (RF)

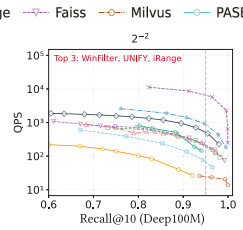


Figure 15: Large-Scale Dataset (RF)

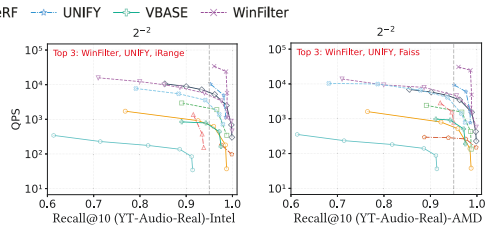


Figure 16: Different Platforms (RF)

5 DISCUSSION

Based on the performance of algorithms across various experimental settings, we evaluate their strengths and weaknesses in different application scenarios, as shown in Table 3. Next, we provide a comprehensive discussion of the advantages and disadvantages of each algorithm.

5.1 Recommendations.

5.1.1 Attribute Filtering. Overall, UNG, NHQ and FilteredDisk -ANN demonstrate strong performance. However, UNG suffers from performance degradation when the attribute count differs between indexing and querying. NHQ, due to its distance fusion design, also experiences performance drops when there is a significant mismatch in attribute numbers between indexing and querying, and its performance is sensitive to attribute distribution.

The two variants of Filtered-DiskANN (FilteredVamana and StitchedVamana) support only single-attribute queries and multi-attribute OR queries. FilteredVamana supports dynamic insertion and disk-based indexing, making it more suitable for extremely large-scale datasets even in memory-constrained environments. However, its disk-based construction significantly increases indexing time, and its performance is highly sensitive to attribute distribution. In contrast, StitchedVamana offers better search performance but performs poorly on 100M-scale datasets and OOD datasets.

CAPS, based on clustering, features low index space overhead. It offers flexible support for varying numbers of attributes and is optimized for low-selectivity attribute scenarios, but its overall performance is relatively moderate. Puck is well-suited for large-scale datasets such as those with 100 million points. It offers fast index construction and low memory consumption in such scenarios.

5.1.2 Range Filtering. WinFilter achieves the best search performance but requires high time and memory resources during index construction. For large-scale datasets (e.g., with 100 million points), parameter tuning—particularly of tree height—is needed to prevent excessive index growth. When QPS and recall are not strict requirements, its basic variant, Vamana-WST, offers a more efficient alternative with significantly lower indexing overhead.

iRange has relatively low time and space costs for index construction. However, its search performance declines significantly with increasing query range. Despite offering threading parameters for indexing, it does not effectively utilize multi-core parallelism in practical scenarios. SeRF incurs low time and space overhead but achieves only moderate overall query performance, with notably weaker results in small-range query scenarios. DSG supports unordered dynamic insertions and range queries, but its index construction is limited to single-threaded execution, resulting in long build times and poor practical applicability.

UNIFY adopts multiple filtering strategies tailored to different query range sizes, avoiding the limitations of a single strategy in specific scenarios. It requires only one index to support all queries, allows dynamic insertions, and features low time and space overhead during index construction, making it a more general-purpose solution.

5.1.3 General Algorithm. All of the following methods support both attribute filtering and range filtering.

ACORN demonstrates average overall performance. Among its variants, ACORN-1 shows significant advantages in indexing speed on 100M-scale datasets, but suffers from high memory consumption and delivers only moderate performance.

Faiss, Milvus and VBASE effectively handle complex filtering by employing versatile "pre-filtering" or "post-filtering" strategies, granting them significant flexibility and broad applicability. This flexibility, however, is often counterbalanced by their relatively modest overall search performance. Relatively speaking, PASE performs less well across various aspects, which indicates that its strategy of loading data pages on-demand is unsuitable for performance-demanding scenarios.

Table 3: Algorithm Recommendation

Application Scenarios	AF						RF					AF / RF				
	UNG	NHQ	Filtered	Stitched	CAPS	Puck	Win	iRange	SeRF	DSG	UNIFY	ACORN	Faiss	Milvus	VBASE	PASE
Performance-Critical	**B**	G	M	G	M	G	G	M	M	M	M	M/M	M/M	M/M	P/P	P/P
Index-Efficient	M	M	M	M	M	G	P	M	G	P	P	P/P	**B**/**B**	G/G	P/M	P/P
Memory-Constrained	M	P	M	M	M	G	P	P	M	P	P	P/P	**B**/**B**	G/M	G/M	G/G
Large-Scale	**B**	G	G	P	P	G	**B**	G	M	M	G	M/P	P/M	P/P	M/M	-
Multi-Modal	**B**	G	G	P	M	M	**B**	G	M	M	G	P/P	P/M	M/M	P/P	P/P
Rare Attribute / Low Range	**B**	M	M	G	G	G	**B**	G	P	G	M	M/M	M/G	P/P	P/P	P/P
Distribution-Robust	G	P	P	G	G	G	-	-	-	-	-	G/-	P/-	**B**/-	P/-	-
General-Purpose	G	G	G	P	P	P	G	P	P	P	G	P	G	G	G	P

Performance Levels: **B** = Best, G = Good , M = Moderate, P = Poor, - = Not Supported; *Performance-Critical*: Requires low latency and high recall; *Index-Efficient*: Emphasizes lightweight index construction; *Memory-Constrained*: Operates under limited memory conditions; *Large-Scale*: For scenarios involving datasets at the scale of 100 million vectors; *Multi-Modal*: For multi-modal dataset scenarios; *Rare Attribute / Low Range*: For filtering with low attribute selectivity or small ranges; *Distribution-Robust*: For scenarios with variable attribute distributions; *General-Purpose*: Balanced performance across varied scenarios.

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 suitable for 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.

In addition, existing attribute filtering algorithms face two major limitations: strict requirements on attribute format and limited support for complex filtering conditions.

For example, in multi-attribute scenarios, some algorithms only support Boolean OR logic (e.g., FilterDiskANN), while others only support Boolean AND logic (e.g., NHQ and CAPS). Although algorithms such as ACORN, UNG, and Puck support both AND and OR logic across multiple attributes, they still struggle to handle complex Boolean expressions that involve nested combinations of conjunctions (AND) and disjunctions (OR). In contrast, while databases and the vector search library Faiss support more expressive Boolean logic, their performance is often suboptimal in queries involving only basic attribute filtering. Therefore, designing algorithms that can support complex filtering expressions while maintaining high query efficiency remains a key challenge to be addressed.

Notably, we find that IVF-based methods are easily extendable to support joint query scenarios of attribute filtering and range filtering, and are compatible with arbitrary combinations of Boolean logic. Our experiments also indicate that IVF-based methods (CAPS, Puck) achieve performance comparable to graph algorithms, while having the lowest time and space overhead, making them highly suitable for large-scale data processing. Therefore, IVF-based hybrid query methods represent a very promising research direction.

Besides, our research indicates that the distribution and selectivity of attributes significantly impact the performance of attribute filtering methods. Therefore, when designing algorithms, efforts should be made to minimize the coupling with specific attribute value distributions, in order to enhance the robustness and generalization ability of the algorithms under varying data conditions.

5.2.2 Range Filtering. Except for DSG and UNIFY, most range filtering algorithms require the dataset to be pre-sorted before building the index. Moreover, existing range filtering algorithms do not support range queries with more than 3 attributes. Designing algorithms that support an arbitrary number of range filterable attributes remains an open research challenge.

In addition,our experiments show that different algorithms exhibit varying strengths under different query ranges. Therefore, designing a general range query algorithm that can adapt to diverse query conditions holds significant research value. For example, UNIFY demonstrates strong generality by dynamically selecting execution strategies based on the query range. Building on this idea, further exploration of more adaptive and high-performance range query algorithms is a promising direction for future research.

Moreover, 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 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, existing algorithms suffer from significant performance degradation on OOD datasets. Exploring how to improve these algorithms to better adapt to more complex query conditions is a worthwhile direction for further research.

6 CONCLUSION

In this paper, we systematically evaluate various hybrid query methods, including multiple algorithms, vector databases, and libraries. We enrich the attribute information of the dataset and design a variety of experimental scenarios, and comprehensively analyze the performance of each algorithm. Experiments show that existing algorithms still have significant shortcomings in scenarios such as large-scale data, complex queries, and resource constraints. There are still significant deficiencies in the flexibility and Boolean logic support of attribute filtering algorithms. Furthermore, the performance of range filtering algorithms remains limited under high-recall requirements and lacks support for multi-attribute queries. Notably, in some high-recall scenarios, some algorithms even perform worse than the baseline method (e.g., Faiss). In the future, it is necessary to further explore the flexibility and robustness of attribute filtering algorithms, while improving the performance of range filtering algorithms is also an important direction. Additionally, exploring algorithms that support a combined and efficient use of attribute and range filtering is also an important research topic.

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] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-based Vector Search: An Experimental Evaluation of the State-of-the-Art. Proc. ACM Manag. Data 3, 1 (2025), 1–31.

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

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

[12] 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.

[13] 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.

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

[15] 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

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

[17] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, and et al. 2025. The Faiss library. (2025). arXiv:2401.08281 [cs.LG] https://arxiv.org/abs/2401.08281

[18] Joshua Engels, Benjamin Landrum, Shangdi Yu, and et al. 2024. Approximate nearest neighbor search with window filters. In Proceedings of the 41st International Conference on Machine Learning (ICML) (ICML'24). JMLR.org, 12469 – 12490.

[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, and et al. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In WWW. ACM, 3406–3416.

[22] Gaurav Gupta, Jonah Yi, Benjamin Coleman, and et al. 2023. CAPS: A Practical Partition Index for Filtered Similarity Search. (2023). arXiv:2308.15014 [cs.IR] https://arxiv.org/abs/2308.15014

[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] 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.

[27] 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.

[28] 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.

[29] Anqi Liang, Pengcheng Zhang, Bin Yao, and et al. 2025. UNIFY: Unified Index for Range Filtered Approximate Nearest Neighbors Search. Proc. VLDB Endow. 18, 4 (2025), 1118–1130. https://doi.org/10.14778/3717755.3717770

[30] 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).

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] Zhencan Peng, Miao Qiao, Wenchao Zhou, and et al. 2025. Dynamic Range-Filtering Approximate Nearest Neighbor Search. Proc. VLDB Endow. 18, 10 (2025), 3256–3268. https://doi.org/10.14778/3748191.3748193

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

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

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

[40] Jayaram Subramanya, Suhas Devvrit, Harsha Vardhan Simhadri, and et al. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In Advances in Neural Information Processing Systems, Vol. 32. Curran Associates, Inc., 13766 – 13776. https://proceedings.neurips.cc/paper_files/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf

[41] 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.

[42] 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.

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

[44] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, and et al. 2023. An Efficient and Robust Framework for Approximate Nearest Neighbor Search with Attribute Constraint. In NeurIPS.

[45] 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.

[46] 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.

[47] Chuangxian Wei, Bin Wu, Sheng Wang, and et al. 2020. AnalyticDB-V: A Hybrid Analytical Engine Towards Query Fusion for Structured and Unstructured Data. Proc. VLDB Endow. 13, 12 (2020), 3152–3165.

[48] 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.

[49] 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.

[50] Qianxi Zhang, Shuotao Xu, Qi Chen, and et al. 2023. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In OSDI. USENIX Association, 377–395.

[51] 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.