Filtered Approximate Nearest Neighbor Search: A Unified Benchmark and Systematic Experimental Study [Experiment, Analysis & Benchmark]

Jiayang Shi
Fudan University, Shanghai, China
jiayangshi22@m.fudan.edu.cn

Yuzheng Cai
Fudan University, Shanghai, China
yuzhengcai21@m.fudan.edu.cn

Weiguo Zheng
Fudan University, Shanghai, China
zhengweiguo@fudan.edu.cn

ABSTRACT

For a given dataset $\mathcal{D}$ and structured label f, the goal of Filtered Approximate Nearest Neighbor Search (FANNS) algorithms is to find top-k points closest to a query that satisfy label constraints, while ensuring both recall and QPS (Queries Per Second). In recent years, many FANNS algorithms have been proposed. However, the lack of a systematic investigation makes it difficult to understand their relative strengths and weaknesses. Additionally, we found that: (1) FANNS algorithms have coupled, dataset-dependent parameters, leading to biased comparisons. (2) Key impact factors are rarely analyzed systematically, leaving unclear when each algorithm performs well. (3) Disparate datasets, workloads, and biased experiment designs make cross-algorithm comparisons unreliable.

Thus, a comprehensive survey and benchmark for FANNS is crucial to achieve the following goals: designing a fair evaluation and clarifying the classification of algorithms, conducting in-depth analysis of their performance, and establishing a unified benchmark. First, we propose a taxonomy (dividing methods into *filter-then-search*, *search-then-filter*, *hybrid-search*) and a systematic evaluation framework, integrating unified parameter tuning and standardized filtering across algorithms to reduce implementation-induced performance variations and reflect core trade-offs. Then, we conduct a comprehensive empirical study to analyze how query difficulty and dataset properties impact performance, evaluating robustness under pressures like filter selectivity, Recall@k, and scalability to clarify each method's strengths. Finally, we establish a standardized benchmark with real-world datasets and open-source related resources to ensure reproducible future research.

1 INTRODUCTION

Recent advances in recommendation systems [17, 24], search engines [18, 43], and AI systems [12, 51] have driven a growing

need to retrieve and process large-scale, multi-modal unstructured data, including text, images, and video. The most popular solution is to unify different modals into high-dimensional embeddings, then apply Approximate Nearest Neighbor Search (ANNS) algorithms to search semantically similar items in high-dimensional spaces [7, 9, 26, 45, 48]. Furthermore, besides solely utilizing semantic similarity as the retrieval criteria, real-world applications [23, 25, 32] also increasingly demand hybrid query processing that integrates semantic search with metadata filtering, such as e-commerce systems [28, 33] retrieving similar products within specified price, or academic search platforms identifying topically relevant publications filtered by publication date and venue. This task combines vector similarity search with structured attribute constraints, and is commonly known as Filtered Approximate Nearest Neighbor Search (FANNS) [3, 14] or constrained Approximate Nearest Neighbor Search [16, 49].

1.1 Background

Filtered Approximate Nearest Neighbor Search (FANNS) is challenging due to the separated label spaces and vector spaces. Naive implementations directly split label filtering and vector retrieval into distinct steps, which apply Boolean filters either pre- or post-ANNS search. However, such intuitive methods suffer from fundamental efficiency limitations. Hybrid approaches remain suboptimal due to substantial computational overhead: systems frequently retrieve and process high-dimensional vectors that satisfy proximity requirements but fail attribute predicates, resulting in unnecessary distance computations and increased query latency [14, 47].

Despite the proliferation of FANNS algorithms, a clear, comparative understanding of their practical performance remains elusive due to three fundamental challenges in the existing works.

Combinatorial Explosion of Algorithmic Configurations. The objective evaluation of algorithms is hindered by complex and coupled parameter spaces. As illustrated in Table 1, most methods expose a relatively large number of tuning parameters. These parameters are often strongly interdependent [50], and their optimal configuration can vary dramatically with changes across datasets. This frequently leads to reported performance reflecting expert-level hyperparameter optimization while baseline methods may receive suboptimal configurations (with default or superficial parameters), obscuring true algorithmic merit. Establishing systematic evaluation protocols that decouple parameter tuning effects from intrinsic algorithmic advantages is therefore essential.

Diverse Characteristics of Queries and Datasets. The precise impact of query characteristics and dataset properties on FANNS performance remains insufficiently characterized. And prior studies [14, 37, 47] frequently neglect systematic analysis of query

Table 1: Dimension of Parameter Space for FANNS Methods

Algorithm	Dim	Algorithm	Dim
Pre-filter Brute Force	0	Post-filter HNSW	2
Post-filter IVFPQ	3	ACORN-γ	3
ACORN-1	2	UNG	4
Filtered DiskANN	3	Stitched DiskANN	4
NHQ-kgraph	9	CAPS	2

difficulty despite performance being governed by multifaceted factors: filter selectivity, predicate complexity, target recall thresholds (*Recall@k*), and degradation patterns under dataset scaling [3, 16]. A comprehensive investigation is needed to understand how different algorithms respond to these varying query- and data-specific pressures, identifying robust approaches across diverse scenarios.
The Hidden Bias and Inconsistency in Evaluation. The experimental setups, datasets, and filter workloads used in existing research [14, 16, 47] are highly fragmented, as different papers often employ varied datasets and experimental scenarios. This inconsistency is compounded by implicit design bias favoring newly proposed algorithms [3, 37], manifested through non-standardized selectivity regimes, where some studies emphasize high-selectivity scenarios while others focus on low-selectivity cases or single-label constraints. The absence of unified benchmarking standards impedes reliable cross-algorithm comparison and obstructs definitive identification of the advantages of these techniques.

1.2 Contributions

To address the challenges above, we conduct a comprehensive benchmark evaluation of FANNS approaches across multiple dimensions. The systematic assessment spans diverse datasets, filter selectivity regimes, and query workloads to characterize fundamental tradeoffs and operational constraints.
A Framework for Fair, Parameter-Aware Evaluation. We introduce a systematic evaluation framework designed to ensure fairness and parameter-aware comparisons, thereby mitigating the evaluation bias often introduced by ad-hoc tuning. We establish an intuitive taxonomy that categorizes methods into three distinct classes: *filter-then-search*, *search-then-filter*, and *hybrid-search*, which serves to elucidate their inherent algorithmic trade-offs. To enable a rigorous and objective comparison, we integrate a unified parameter tuning algorithm and a standardized filtering implementation, supporting both pre-computed bitsets and post-filtering strategies, across all evaluated methods. We evaluated 9 algorithms across small subsets of 6 datasets, testing over $41,000$ parameter combinations to build small indices and perform hybrid search. From these, we selected approximately $1,300$ representative parameter sets to construct full indexes on the entire datasets. This methodological rigor, which includes a comprehensive exploration of parameter combinations and index configurations, minimizes performance variance attributable to implementation-specific optimizations. Consequently, our benchmarking results isolate and accurately reflect the fundamental performance characteristics and core trade-offs of each algorithmic paradigm.
In-Depth Comparison Across Diverse Query and Dataset Characteristics. We conduct a comprehensive empirical study

characterizing how query difficulty and dataset properties influence search performance. Our experimental design systematically evaluates algorithmic robustness across controlled operational pressures, including filter selectivity gradients, recall thresholds (Recall@k), and scalability limits. For example, we test performance under data scalability pressures by progressively increasing the dataset size from 5 thousand to 5 million items in Section 5.8, which reveals the practical robustness and specific strengths of each approach. This rigorous assessment reveals comparative strengths and operational boundaries of each approach under realistic deployment conditions.
A Unified and Reproducible Benchmark. To address the fragmentation and incomparability of existing experimental methodologies, we establish a standardized benchmark. This benchmark integrates multiple large-scale, real-world datasets with diverse characteristics and authentic labeling structures, such as YFCC[46] and YouTube[1], and defines a comprehensive suite of filter workloads. Our open-source release of all datasets, evaluation scripts, and unified algorithm implementations provides a foundational resource for transparent and reproducible evaluation, ensuring that future research advances are built upon consistent and equitable experimental foundations.

In summary, we make the following contributions in this paper.

- To enable objective benchmarking, we present a novel evaluation framework that alleviates parameter tuning bias, thereby providing a clear analysis of inherent algorithmic trade-offs.
- We conduct a comprehensive experimental study to reveal the practical strengths and operational boundaries of different algorithms across diverse query and data scenarios.
- We present and open-source a unified benchmark featuring standardized datasets and evaluation scripts, designed to foster transparency, reproducibility, and rigorous comparative analysis.

2 PRELIMINARY

2.1 Problem Definition

We present the definitions of approximate nearest neighbor search and filtered approximate nearest neighbor search.

Definition 2.1 (Approximate Nearest Neighbor Search, ANNS). Let $\mathcal{D} \subset \mathbb{R}^d$ be a base dataset of n vectors and let $v_q \in \mathbb{R}^d$ be a query vector. Given an integer $k \geq 1$, the approximate k-nearest neighbor search algorithm returns an approximate k-nearest neighbor set (k-NN) of size k, $\mathcal{R}(v_q) = \{x_1, \ldots, x_k\} \subseteq \mathcal{D}$, to minimize the distance of $\delta(v_q, v_i)$ for any $v_i \in \mathcal{R}(v_q)$ while reducing latency.

We now extend it to define FANNS, where each vector $v_i \in \mathcal{D}$ is associated with a set of labels f_i. A filtered query must find vectors that are close in distance and satisfy a given label constraint [3, 14].

Definition 2.2 (Filtered Approximate Nearest Neighbour Search, FANNS). Let $\mathcal{D} \subset \mathbb{R}^d$ be a base dataset of n vectors, where each vector $v_i \in \mathcal{D}$ is associated with a label set f_i. Let $v_q \in \mathbb{R}^d$ be a query vector with its own label set f_q. Given an integer $k \geq 1$ and a filter constraint $\mathcal{S}$, the filtered approximate k-nearest neighbour search algorithm aims to find the approximate k-NN set of size k, $\mathcal{R}_\mathcal{S}(v_q) = \{x_1, \ldots, x_k\} \subseteq \mathcal{D}$, such that two conditions are met:

(1) **Filter Satisfaction**: For every vector $x_j \in \mathcal{R}_\mathcal{S}(v_q)$, its corresponding label set f_j must satisfy the constraint $\mathcal{S}$ with respect to the query's label set f_q, denoted as $f_j \mid_\mathcal{S} f_q$.

Table 2: Comparison of support for different logical predicates across various algorithms. Abbreviations: CONT (Containment), OVER (Overlap), EQ (Equality), FIXED-EQ (Fixed Length Equality), COMB (Combination). COMB refers to the ability to handle complex logical expressions involving multiple predicates.

METHOD	CONT	OVER	EQ	FIXED-EQ	COMB	Method Paradigm
NHQ [47]	✗	✗	✗	✓	✗	Hybrid-Search
FILTERED-DISKANN [14]	✓	✓	✓	✓	✓	Hybrid-Search
STITCHED-DISKANN [14]	✓	✓	✓	✓	✓	Hybrid-Search
ACORN-1 [37]	✓	✓	✓	✓	✓	Filter-then-Search
ACORN-γ [37]	✓	✓	✓	✓	✓	Filter-then-Search
CAPS [16]	✗	✗	✗	✓	✗	Hybrid-Search
UNG [3]	✓	✓	✓	✓	✗	Filter-then-Search
BRUTE-FORCE [3]	✓	✓	✓	✓	✓	Filter-then-Search
Post-filter HNSW [29]	✓	✓	✓	✓	✓	Search-then-Filter
Post-filter IVFPQ [19]	✓	✓	✓	✓	✓	Search-then-Filter

(2) **Approximate Proximity**: The set $\mathcal{R}_S(v_q)$ contains vectors that are an approximation of $Gt_S(v_q)$, that is, the true k-nearest neighbors to v_q among all vectors in $\mathcal{D}$ that satisfy the filter constraint S.

Next, we present four filter constraints S commonly used in recent FANNS studies, followed by an illustrating example.

Definition 2.3 (Containment). A vector $x_j \in \mathcal{R}_S(v_q)$ with label set f_j satisfies $f_j \mid_S f_q$ iff $f_q \subseteq f_j$.

Definition 2.4 (Overlap). A vector $x_j \in \mathcal{R}_S(v_q)$ with label set f_j satisfies $f_j \mid_S f_q$ if and only if $f_q \cap f_x \neq \emptyset$.

Definition 2.5 (Equality). A vector $x_j \in \mathcal{R}_S(v_q)$ with label set f_j satisfies $f_j \mid_S f_q$ iff $f_q = f_j$.

Definition 2.6 (Fixed-Length Equality). All vectors in dataset $\mathcal{D}$ must have the same numbers of labels, i.e., $|f_i| = |f_j|$ for $x_i, x_j \in \mathcal{D}$. And the filter constraint $f_j \mid_S f_q$ is satisfied if and only if $f_q = f_j$.

Example 2.7. Figure 1 illustrates how filtering scenarios affect nearest-neighbor results for identical queries. Without filtering, v_5 is the 1-NN (the nearest neighbour to query). Under containment constraints that desire vectors with label set containing query labels (i.e., $f_q \subseteq f_i$), v_4, v_5, and v_6 are excluded, as their label set does not contain f_q, making v_3 the 1-NN. Similarly, For equality constraints that require $f_i = f_q$, only v_3 and v_7 meet the filter condition, leaving v_3 as 1-NN. Under overlap constraints that desire vectors with labels overlapped with query labels (i.e., $f_i \cap f_q \neq \emptyset$), only v_4 is excluded ($f_4 \cap f_q = \emptyset$), restoring v_5 as 1-NN.

Notably, beyond these filter constraints on categorical labels, recently *range filters* also arise, where each vector v_i has a single numerical attribute like a timestamp or a price. A query consists of a vector v_q and a continuous range $[l, r]$, aiming to find the k-nearest neighbors to v_q whose attributes fall within this range. Though there are a few specialized solutions [27, 52, 53], we find that they are merely compatible with the majority of FANNS algorithms and thus are not covered in this survey.

2.2 Evaluation Metrics

To assess the performance of FANNS algorithms, we evaluate both search accuracy and operational efficiency. These aspects are captured by the following standard metrics.

Definition 2.8 (Recall). Recall measures the quality of the search result by quantifying the fraction of true nearest neighbors that are successfully retrieved. For a given FANNS query v_q and a target number k, the recall is defined as:

$$\text{Recall@k} = \frac{|\mathcal{R}_S(v_q) \cap Gt_S(v_q)|}{|Gt_S(v_q)|},$$

where $Gt_S(v_q)$ is the ground truth set of the FANNS query and $\mathcal{R}_S(v_q)$ is the result set returned by the algorithm.

To evaluate search efficiency, queries per second are commonly used to report query throughput.

Definition 2.9 (Queries Per Second, QPS). QPS is calculated as the total number of queries processed divided by the total time elapsed, i.e., how many queries the system can handle per unit time.

The performance of a FANNS algorithm, in terms of both recall and QPS, is heavily influenced by the difficulty of the query itself. The filter selectivity is a key factor influencing its difficulty.

Definition 2.10 (Filter Selectivity). Filter selectivity, denoted by $\sigma_S(f_q)$, is the fraction of the dataset that satisfies the given filter constraint S with respect to a query's label set f_q:

$$\sigma_S(f_q) = \frac{|\{v_i \in \mathcal{D} \mid f_i \mid_S f_q\}|}{|\mathcal{D}|}.$$

3 FANNS ALGORITHMS

We categorize FANNS algorithms into three distinct classes according to the execution stage of the filtering process: filter-then-search, search-then-filter, and hybrid-search. Each category exhibits different search workflows. Subsequently, we will explain all specific algorithms for each class, and provide a detailed analysis of their variations in index construction and query processing. Table 2 summarizes the support of various search algorithms for five key logical predicates, as well as their respective filter type classifications.

3.1 Filter-Then-Search ANNS Algorithms

Filter-then-search algorithms explicitly or implicitly filter the entire dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ to extract vectors satisfying specified label constraints. This process generates a filtered subset $\mathcal{D}_f = \{\mathbf{x}_{k_1}, \mathbf{x}_{k_2}, \ldots, \mathbf{x}_{k_m}\}$ where $\mathcal{D}_f \subseteq \mathcal{D}$. Subsequent nearest neighbor search operates exclusively within this filtered subset $\mathcal{D}_f$, requiring no additional constraint verification during query processing. The
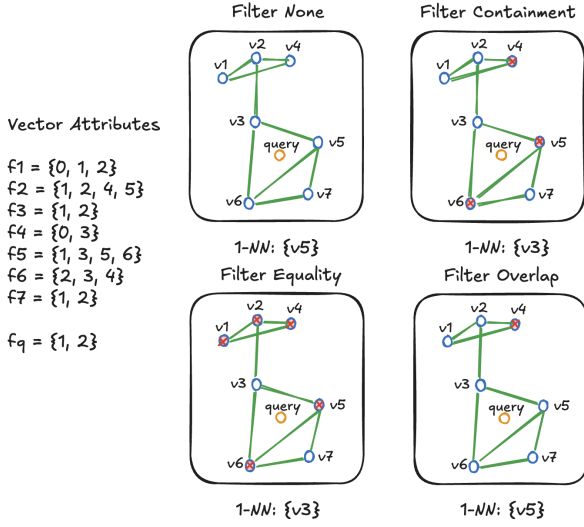
Figure 1: Visualization of FANNS: for 1-NN search (i.e., searching for the point closest to the query) under four scenarios (None, Containment, Equality, Overlap), results vary significantly across different scenarios.

Figure 2: Example of ACORN-1: it illustrates that ACORN increases the breadth of search by finding two-hop neighbors.

search procedure reduces to solving a conventional ANNS problem over the constrained dataset.

A critical implementation combines pre-filtering with vector quantization [8]. This approach first utilizes a fast bitset to identify the IDs of all points that satisfy the metadata constraints. Then, a quantization algorithm is used to rapidly compare the approximate distances between these candidate points and the query vector. Finally, a small subset of the most promising candidates is selected for a reranking step using their full-precision vector representations.

Creating the filter map relies on a pre-computed inverted index, where each label maps to a bitset indicating all vectors that possess it. Given a query label set f_q, the filter map for *Containment Scenario* is built by performing a bitwise AND operation on all the bitsets corresponding to the labels in f_q. Conversely, the filter map for *Overlap Scenario* is built by performing a bitwise OR operation on all these bitsets. For *Equality Scenario*, we can first perform a bitwise AND operation identical to the *Containment Scenario*, followed by iterating these candidates to perform the more expensive check for exact set equality. Given the machine word size w, such implementation achieves a time complexity of $O(|\overline{f_q}| \cdot N/w)$ for each query with an average of $|\overline{f_q}|$ bitset operations on a dataset of size N. To store the bitset of each label, it requires $O(|f_{total}| \cdot N/w)$ space, where $|f_{total}|$ is the possible number of labels.

Example 3.1. As shown in Figure 1, for a query $L_q = \{1, 2\}$ against a database of 7 items, we first retrieve the pre-computed bitsets. The set of items possessing label 1 is $\{v_1, v_2, v_3, v_5, v_7\}$, which corresponds to the bitset $\mathcal{B}_1 = [1, 1, 1, 0, 1, 0, 1]$. Similarly, the items with label 2 are $\{v_1, v_2, v_3, v_6, v_7\}$, yielding the bitset $\mathcal{B}_2 = [1, 1, 1, 0, 0, 1, 1]$. To find items that contain both labels (AND logic), we perform a bitwise AND: $\mathcal{B}_1 \wedge \mathcal{B}_2 = [1, 1, 1, 0, 0, 0, 1]$. To find items that contain either label (OR logic), we perform a bitwise OR: $\mathcal{B}_1 \vee \mathcal{B}_2 = [1, 1, 1, 0, 1, 1, 1]$.
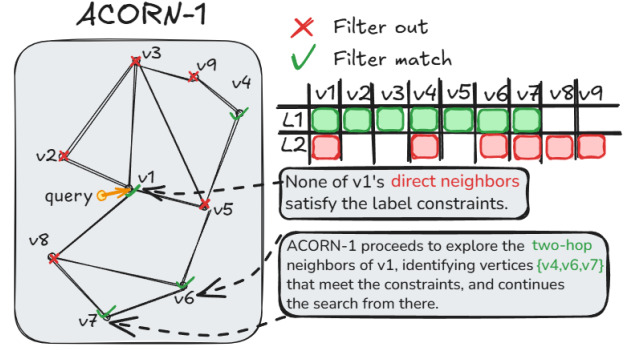
After pre-filtering with bitwise operations, the vector quantization is used for fast distance computing. It has evolved from foundational techniques like Product Quantization (PQ) [20] and its Optimized Product Quantization (OPQ) [13] variant to more recent algorithms like RaBitQ [11] and Extended-RaBitQ [10], which have been proven to have superior theoretical error bounds.

Pre-filter Brute Force. We implement a baseline algorithm that first filters the dataset and then applies brute-force search. The filter map is used to retrieve the subset of database ($\mathcal{D}_f \subseteq \mathcal{D}$) that satisfies the filter constraints. Then, it performs a brute-force, full-precision search within filtered vector set ($\mathcal{D}_f$) and returns the top-k results.

ACORN-γ [37]. The ACORN-γ algorithm constructs its without any label information in advance. This allows for flexible data updates, as labels can be assigned or modified after the index is built. The core construction idea is to create a denser graph by expanding each node's degree by a factor γ, which is determined based on an estimated selectivity. A specialized predicate-agnostic pruning technique is then applied to control index size. At query time, ACORN-γ first performs a full-dataset scan to generate a bitset of all vectors satisfying the filter constraint (which can be as complex as a regular expression). The subsequent graph traversal is then guided by this bitset. When expanding its candidate lists, it decompresses the neighbor information and checks filtering conditions, which broadens the search's scope. This algorithm can be combined with mainstream indexes like HNSW and DiskANN and performs well with high selectivity, and many vector search engines have added support for it, like ElasticSearch [8]. Due to this initial dataset-wide filtering step, we classify ACORN-γ as a filter-then-search approach.

ACORN-1 [37]. In contrast to ACORN-γ, ACORN-1(see Figure 2) offers a lightweight alternative that aims to approximate the search performance of ACORN-γ while significantly reducing index construction time and memory cost. The key difference lies in its approach to neighbor expansion. Instead of creating a denser graph during construction, ACORN-1 builds a standard, un-pruned HNSW index. During the greedy search, then visiting a node v, ACORN-1 dynamically expands the search scope to consider not only its direct (one-hop) neighbors but also its two-hop neighbors. This expanded
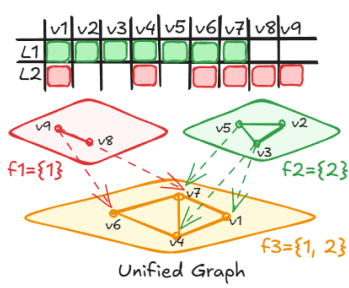
Figure 3: Example of UNG. The dataset is partitioned into disjoint groups by label sets: f_1={1}, f_2={2}, and f_3={1,2}. Each group has its own intra-group graph with no overlap of vectors across groups. Inter-group connections follow the label-navigating graph and respect containment (e.g., {1}→{1,2} and {2}→{1,2}), so any edge (v_i, v_j) implies $f_i \subseteq f_j$.

Figure 4: Example of Post Filter HNSW.

Figure 5: Example of Stitched DiskANN.

candidate set is then filtered according to the query predicate before the best neighbors are pushed into the priority queue.

UNG [3]. The Unified Navigating Graph (UNG) algorithm pre-compiles label information into a label navigating graph structure during offline construction, which encodes the containment relationships of different label sets (see Figure 3). It groups vectors by their labels, builds intra-group graphs, and adds inter-group edges based on a label navigation graph (LNG). This design guarantees that if an edge (v_i, v_j) exists, their labels satisfy a containment relationship ($f_i \subseteq f_j$). By locating the entry labels that follow the filter constraints, greedy traversal on the proximity graph never requires further label checking, making UNG a highly efficient filter-then-search method for containment and equality predicates. For equality scenarios, this index degenerates into a search on multiple independent small graphs, and for overlap scenarios, multiple rounds of searching are required, followed by merging the results to ensure recall in their implementation. However, its performance suffers significantly on queries with more complex constraints, such as label overlap scenarios, as multiple rounds of search are performed starting from different entries.

3.2 Search-Then-Filter ANNS Algorithms

Search-then-filter algorithms perform an unconstrained approximate nearest neighbor search over the entire dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, $\mathcal{R} = \{\mathbf{x}_{r_1}, \mathbf{x}_{r_2}, \ldots, \mathbf{x}_{r_l}\}$ ordered by their distance to the query, where typically $l \gg k$ and k denotes the number of search targets (i.e., Recall@k). This approach provides native compatibility with most existing ANNS algorithms. After performing the traditional ANNS search, the constraint verification proceeds as follows:

- If $\mathcal{R}$ contains at least k vectors satisfying the label constraints, the top-k valid results are returned immediately.
- Otherwise, the search scope l is iteratively expanded to retrieve more candidates, repeating the verification process until k valid results are obtained.

Post-filter HNSW [29]. We implement this baseline by combining the standard HNSW algorithm with a post-filtering strategy.(see Figure 4) First, a conventional HNSW index is built on the entire dataset $\mathcal{D}$ without considering any label information. During a
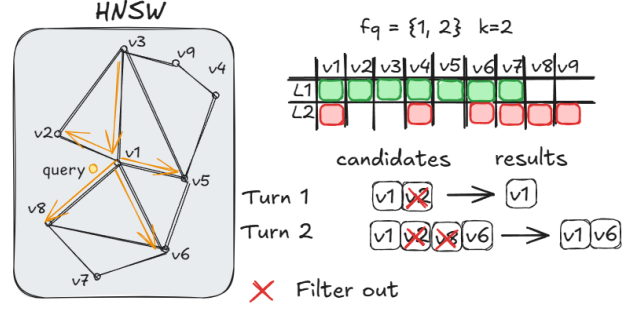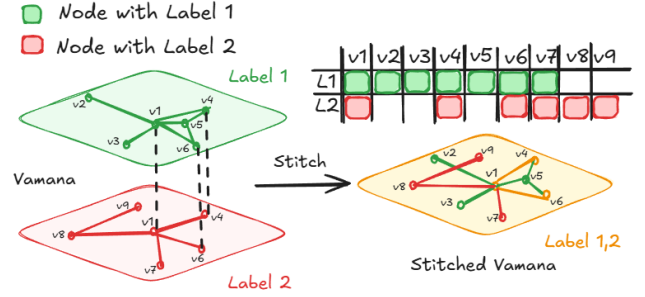
query, we perform an initial ANNS search on this index with a search parameter l that controls the size of results. This result queue is then scanned to check for filter satisfaction. If at least k valid results are found, the top-k among them are returned. If the count is insufficient, the search is re-issued with an expanded scope (e.g., by doubling l) and the process is repeated until k valid neighbors are collected or a maximum search limit is reached.

Post-filter IVF-PQ [19]. Similarly, we implement another baseline using the Inverted File with Product Quantization (IVF-PQ) index. The post-filtering mechanism remains the same, but the underlying search scope is controlled by the *nprobe* parameter, which specifies the number of inverted lists to visit.

3.3 Hybrid-Search ANNS Algorithms

Hybrid-search algorithms integrate label constraints directly into the search process, avoiding explicit pre-filtering or post-filtering stages. This approach typically employs some strategies, such as constraint neighbor expansion or distance fusion. They may build upon foundational ANNS methods, such as Filtered-DiskANN [14] that extends DiskANN [45] by deeply integrating label constraints into both index construction and query processing.

Filtered-DiskANN [14]. This algorithm is based on the Vamana graph, the core graph index of DiskANN [44], but modifies both the construction and search processes to be label-aware. During index construction, when a new point is inserted, only neighbors that share at least one common label with the new point (i.e., $f_i \cap f_q \neq \emptyset$) are added to the candidate queue, and edges are then established with the candidates found along this search path. Finally, a label-aware pruning strategy is applied to refine the connections. We note that the original implementation [42] primarily supports singleton
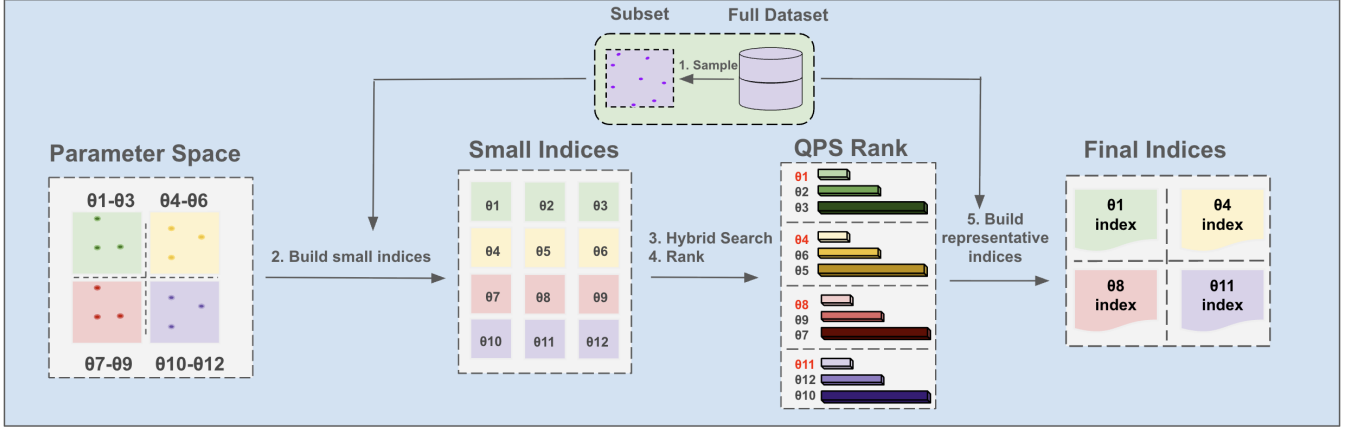
Figure 6: Illustration of Parameter Tuning Algorithm. The algorithm proceeds as follows: 1) randomly sample some points from the dataset as a subset (line 5); 2) traverse the parameter space and build small indices (line 8); 3) run hybrid searches for each small index (line 10); 4) perform parameter ranking in each uniformly divided subspace of the parameter space as detailed in line 14.; 5) select one representative parameter from each subspace and construct full-scale indices (line 16).

query label sets ($|f_q| = 1$). To accommodate more general cases ($|f_q| > 1$) and more scenarios, we extend the algorithm by adding an extra filtering step: after a node is popped from the candidate queue and before it is placed into the final result set, we perform a label check. Because this approach continually performs distance computations with label check throughout the search, we classify it as a hybrid-search algorithm.

Stitched-DiskANN [14]. In contrast to Filtered-DiskANN's graph approach, Stitched-DiskANN (see Figure 5) adopts a "partition-then-stitch" construction strategy. It first partitions the dataset by creating a separate Vamana subgraph [44] for each individual label in the entire label set $\mathcal{L}$. During the construction of each Vamana subgraph, a point is connected to other points whose labels intersect with its own; this design makes its search process highly suitable, as it only needs to expand neighbors that have label intersections with the query and ensures connectivity. These initially disconnected subgraphs are then "stitched" together into a single cohesive index. The stitching process leverages points that carry multiple labels, since these points naturally appear in multiple subgraphs, they serve as crucial bridge nodes that ensure connectivity.

NHQ [47]. NHQ is a hybrid-search algorithm that builds upon foundational graph-based ANNS methods, such as kgraph [5] and NSW [2, 22], and introduces modifications to better adapt to filtering scenarios. It integrates label constraints into both the graph construction and query process by fusing label-based distances with vector distance metrics. Specifically, NHQ defines a combined weight $wt(v_i, v_j)$ using the Hamming distance of the labels between two nodes and the vector distance $\delta(v_i, v_j)$, creating a weighted fusion of these two metrics. For graph construction, NHQ employs a Navigable PG (NPG) strategy, where edges are proactively established between points in diverse directions. This ensures better navigability during the search process, allowing queries to traverse the graph efficiently. In experimental evaluations, we use the NHQ-NPG-kgraph implementation, which incorporates these strategies for both index construction and query execution.

CAPS [16]. CAPS algorithm constructs a two-level index by first partitioning the dataset into coarse spatial clusters using k-means. Each cluster is then further subdivided into a hierarchy of fine-grained, label-based sub-clusters. This is done by iteratively creating sub-clusters for the most frequent label among the remaining vectors, up to a specified number of sub-clusters h. At query time, CAPS first identifies the nearest coarse cluster centroid. It then sequentially probes the second-level sub-clusters within that cluster in their order of creation to find matching results.

4 PARAMETER TUNING

Existing studies demonstrate that vector databases exhibit a high-dimensional parameter space where configurations are highly interdependent [50]. This strong parameter coupling prevents independent optimization of individual parameters. Furthermore, tuning constitutes a complex two-objective optimization problem (maximizing both QPS and recall), making it challenging to identify parameter configurations that yield optimal trade-offs in the QPS-Recall space.

To address these challenges, we propose a parameter tuning approach as shown in Figure 6, which is designed for:

- **Coverage:** Comprehensive exploration of the parameter space.
- **Balance:** Optimal trade-offs via the Pareto frontier.
- **Robustness:** Near-optimal parameters for each specific context.

Algorithm 1 begins by partitioning the entire parameter space Ω into smaller, manageable subspaces (line 1). This strategy ensures a broad exploration of different parameter regions. To accelerate the evaluation process, we perform searches on a randomly sampled subset of the dataset $\mathcal{D}'$ (line 5). For each parameter configuration θ within a subspace, we evaluate its performance across all defined search scenarios and compute the average QPS and Recall (line 11).

A key step in our method is to normalize the comparison of different parameter sets. We evaluate how efficiently a configuration can achieve specific recall targets. We define a set of fixed recall levels (e.g., 0.8, 0.9, 0.95) (line 12) and use linear interpolation to

Table 3: Statistics of Datasets

Dataset	Type	Dim	Number of Vectors	Original Labels	Application
arXiv	text	768	132,678	year, month, task, etc	Academic retrieval
TripClick	text	768	1,055,976	clinical area	Health web search
LAION1M	image	512	1,000,448	entities, locations, etc	Image retrieval
YFCC	image+audio	192	1,000,000	classes	Image retrieval
YouTube Audio	audio	128	5,000,000	classes	Audio retrieval
YouTube Video	audio	1024	1,000,000	classes	Video retrieval

Algorithm 1 Parameter Tuning Algorithm

Require: Algorithm $\mathcal{A}$, Parameter space Ω, Dataset $\mathcal{D}$, Queryset Q
Ensure: A set of representative parameters
1: Divide the parameter space Ω into subspaces $\{\Omega_1, \Omega_2, \ldots, \Omega_n\}$.
2: $Result \leftarrow \emptyset$
3: **for** each subspace $\Omega_i \in \{\Omega_1, \Omega_2, \ldots, \Omega_n\}$ **do**
4:     Initialize $\Theta_i \leftarrow \emptyset$
5:     Generate a sampled dataset $\mathcal{D}' \leftarrow \text{RandomSample}(\mathcal{D})$
6:     **for** each parameter set $\theta \in \Omega_i$ **do**
7:         Initialize $\{qps_s, recall_s\} \leftarrow \emptyset$ for all scenarios $s \in \mathcal{S}$
8:         $\mathcal{I} \leftarrow \text{ConstructIndex}(\mathcal{A}, \theta, \mathcal{D})$
9:         **for** each scenario $s \in \mathcal{S}$ **do**
10:             $\{qps_s, recall_s\} \leftarrow \text{PerformSearch}(\mathcal{A}, \mathcal{I}, s, Q)$
11:         Average metrics: $\{\overline{qps}, \overline{recall}\} \leftarrow \text{avg}_s(\{qps_s, recall_s\})$
12:         Define recall targets $\mathcal{R} \leftarrow [recall_1, recall_2, recall_3]$
13:         $qps_1, qps_2, qps_3 \leftarrow \text{Interpolation}(\{\overline{qps}, \overline{recall}\}, \mathcal{R})$
14:         Rank θ based on qps_1, qps_2, qps_3: r_θ
15:         $\Theta_i \leftarrow \Theta_i \cup \{(\theta, r_\theta)\}$
16:     $\theta^* \leftarrow \arg\min_{\theta \in \Theta_i} r_\theta$
17:     $Result \leftarrow Result \cup \{\theta^*\}$
18: **return** $Result$

---

estimate the QPS that each configuration θ would achieve at these exact recall values (line 13). Each configuration is then assigned a rank based on its interpolated QPS values (line 14). Finally, for each subspace, we select the parameter configuration with the best overall rank as the representative for that region.

5 EXPERIMENTS

In this section, we systematically evaluate diverse real-world workloads within a unified benchmark, thoroughly comparing FANNS algorithms under parameter-aware settings.

5.1 Experimental Setup

Scenarios. As presented in Section 2.1, we cover four filtering scenarios to comprehensively evaluate existing FANNS solutions, including *Containment, Overlap, Equality,* and *Fixed-Length Equality*. The majority of methods, such as ACORN [37] and Post-Filter HNSW [29], natively support the first three scenarios (Containment, Equality, and Overlap), while others, like FilteredDiskANN [14], can be adapted with minor modifications. In contrast, a few algorithms, such as NHQ and CAPS [16], are specifically designed for the more restrictive *Fixed-Length Equality Scenario*, which is a specialized subset of the broader *Equality Scenario*.

Datasets. Table 3 summarises 6 real-world datasets used in the experiments. All selected datasets include real-world labels. Some are

provided directly in vector format, while others contain raw images or text. For the latter, we generated embeddings using pretrained models. Each dataset will be described in detail below:

- **arXiv** [30]. We use the same dataset as [3]: 132,687 papers with methodology embeddings from [35]. Label-equality uses 26 distinct labels (12 months + 14 years). Other scenarios use attributes: task (1678 values), method (890), dataset (1637), with possible multi-values per vector.
- **TripClick** [39] A large-scale health information retrieval dataset, comprising click logs from the Trip Database health web search engine. The dataset includes approximately 1 million user interactions with medical field, clinical area list and publication date, etc. We selected the 27 most frequent labels and categorized all the remaining ones as others.
- **LAION1M** [40] A large-scale, publicly available dataset of 400 million CLIP-filtered[38] image-text pairs, where the natural language text captions serve as descriptive labels for the images. We only selected the first 1000448 items from 4 million embeddings.
- **YFCC** [46]. A standard CV dataset with 100M image/video embeddings in total. Labels include metadata (identifiers, tags, dates, etc). We use precomputed embeddings and formatted labels from [41]. We randomly selected 1M points from the dataset.
- **YouTube-Audio** and **YouTube-Video** [1] A large-scale benchmark of millions of videos annotated with thousands of machine-generated topical entities, providing distinct, pre-computed visual and audio features extracted at one-second intervals. The two datasets share the same attributes.

Figure 7 presents the label distribution statistics across all evaluated real-world datasets. The top row shows that the label ID frequencies typically follow a long-tailed distribution. The middle row indicates the number of labels associated with the data points. The bottom row, which illustrates the group size (the number of items sharing an identical set of labels), also demonstrates a skewed distribution. We ensure that our query labels are sampled to reflect these inherent distributions of the base labels by random sampling, thereby simulating a realistic retrieval environment.

Metrics. We employ the *QPS* (Queries Per Second) and *Recall@k* metrics defined in Section 2.2. Unless otherwise specified, $k = 10$ is used as the default top-k value. For all experiments, we ensure label selectivity defined in Section 2.2 thresholds guarantee at least k ground-truth results per query.

Experiment Environment. Programs are implemented in C++ and compiled with -Ofast Optimization. All the experiments are executed on a Linux server with Intel(R) E5-2596v4 CPU @2.2GHz and 220GB of RAM. For all algorithms, we use 16 threads to construct the index. To process ANNS queries, we use 16 threads for
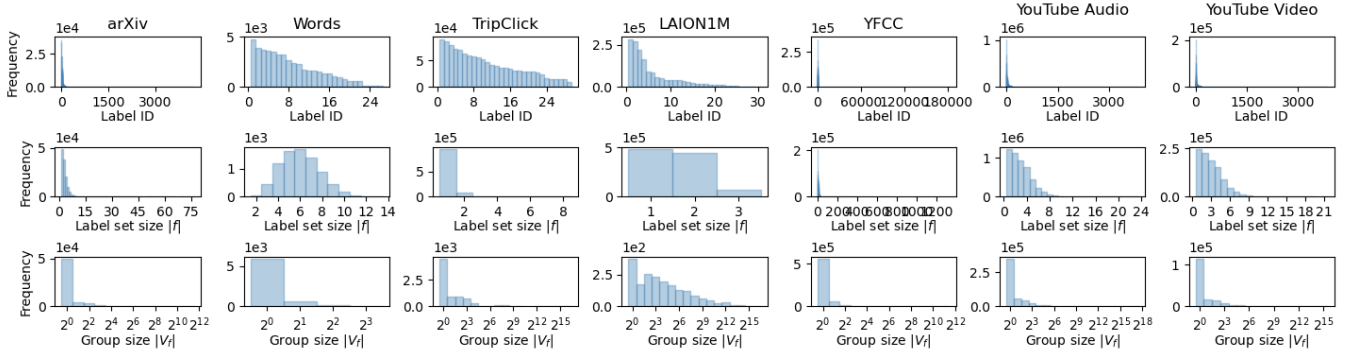
Figure 7: Distributions of base label, size of base label sets, and the number of groups.

Table 4: Base Label Requirements for Index Construction

Base Label Requirement	Algorithms
Requires Base Labels	NHQ† [47] CAPS [16] UNG [3] Filtered-DiskANN† [14] Stitched-DiskANN† [14] Brute-Force [3]
Require No Base Labels	ACORN-1 [37] ACORN-γ [37] Post-Filter HNSW [29] Post-Filter IVFPQ [19]

searching. Furthermore, for single-threaded algorithms [16, 47], we have added a multi-threaded implementation. The multithreading for all algorithms is achieved using the C++ OpenMP library [34]. Due to the random access nature of node traversal [4, 9, 29, 31], implementing parallelization within graph-based algorithms is not straightforward. Therefore, we only apply simple parallelization across different queries, similar to [3, 14]. We have implemented bitset calculation for Acorn as detailed in Section 3.1, which is significantly faster than its brute-force bitset calculation [36] and previous implementations [3]. This bitset is also applied to the post-filtering process in Faiss-HNSW and Faiss-IVFPQ [6]. Additionally, we have implemented multi-label search for FilterDiskANN and StitchedDiskANN [14]. We did not alter the graph construction process; instead, we check if the constraints for all labels are met when populating the result list.

5.2 Performance of Constructing Index

Before evaluating online search performance, we first measure offline index construction time and analyze parameters affecting graph-building efficiency. Since our largest dataset is limited to 500M vectors[1], the index construction cost critically determines algorithmic scalability to larger datasets. We also evaluate whether index building requires *base label* information, which impacts update-friendliness when base labels change.

In Table 4, algorithms marked with $\dagger$ require base labels during initial index construction but support efficient incremental updates.
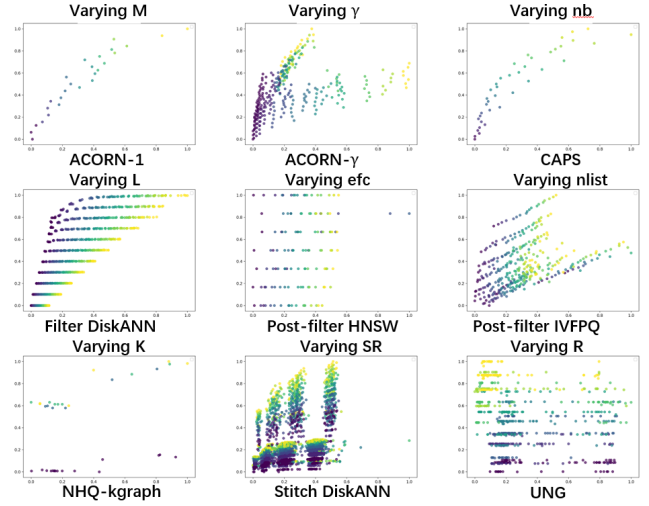

Figure 8: Scatter plots showing the effect of varying parameters on index time and size for different algorithms. Each point represents a specific parameter configuration, normalized to the [0, 1] range.

For these methods, small-scale modifications (vector/label insertions, deletions, or updates) can be handled with minimal impact on overall index structure or search performance.

Figure 8 illustrates the impact of varying parameters on the indexing time and index size for different algorithms. Each scatter plot corresponds to a specific algorithm, with the horizontal axis representing the normalized indexing time and the vertical axis representing the normalized index size. Different parameter values are encoded by the colors of the points. Notably, while each algorithm may have multiple tunable parameters, only one representative parameter was selected for visualization in each scatter plot.

5.3 Overall Query Performance

Our evaluation is conducted on 6 real-world datasets spanning diverse application domains, as detailed in Section 5.1. We leverage the original labels provided with each dataset, which are preprocessed by mapping them to a dense, consecutive integer range. To ensure a fair and consistent evaluation across our three experimental scenarios, the query sets were curated so that each query has at least 10 ground-truth items that fulfill the specified label constraint.
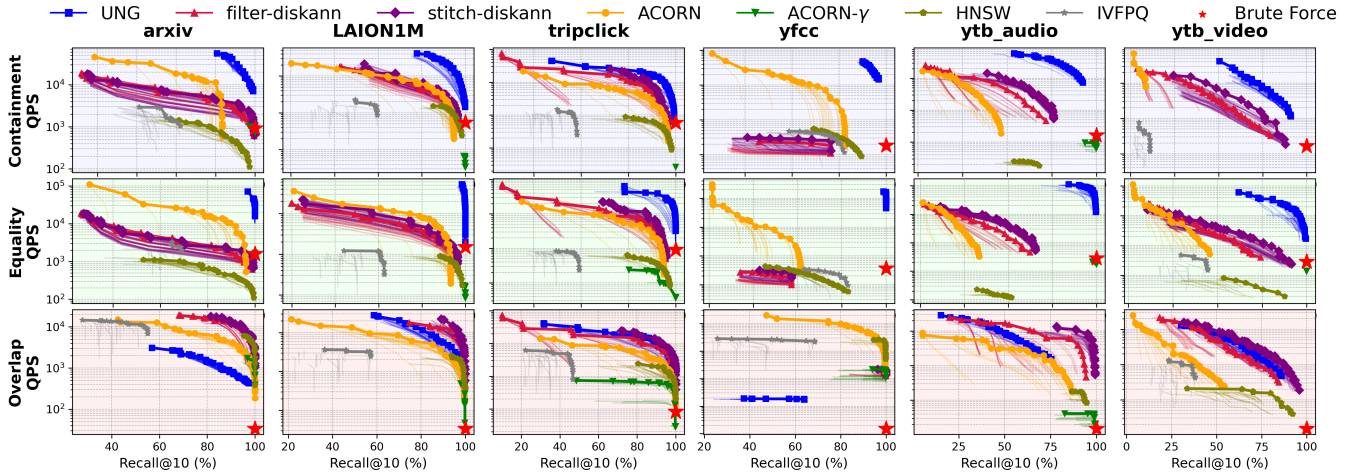
Figure 9: Query performance on 6 real-world datasets.

Figure 9 shows the performance of all algorithms across various datasets, which have significant differences in dimensionality and the number of labels. The QPS-recall curve for each parameter set is represented by a light-colored line. For the points on the Pareto Frontier, they are bolded and shown in a darker color. Unless otherwise specified, this convention applies to all subsequent figures. Accordingly, the algorithms exhibit varied performance on these datasets. Overall, `UNG` is well-suited for the containment and equality scenarios due to its unique filter-then-search design. `DiskANN` performs well in the overlap scenario while maintaining a good balance in others. The specific factors influencing these performance differences will be discussed in detail in the following experimental sections; this section serves as a brief overview.

Figure 10: Result of Fixed Length Equality Scenario.

5.4 Effect of Fixed Length Equality

In previous studies [16, 47], the Fixed Length Equality is a commonly used scenario. For this experiment, we selected two representative datasets, arXiv [30] and YFCC [46]. Since this scenario is the simplest among the four, all algorithms can handle it without modifying their original implementations. Therefore, this experiment involves all the algorithms. As such structured labels are not part of the original datasets, we followed the implementation in NHQ paper [47] to generate synthetic data with a fixed length of 4. Each position in the label has three possible values, chosen with equal probability. Consequently, there are 3^4 possible combinations, and the selectivity of each combination is 1/81.

Figure 10 shows the performance of 10 algorithms in this experiment. Each algorithm has several optimal parameter combinations. In this scenario, most algorithms are capable of achieving high recall values by adjusting their search parameters. Among all tested methods, `UNG` and `CAPS` consistently deliver the highest QPS, surpassing 10^5 on the YFCC dataset at high recall levels. The performance on the YFCC dataset shows a clear stratification of the algorithms, with distinct performance tiers. On the smaller, lower-dimensional arXiV dataset, most algorithms exhibit a noticeable improvement in QPS. This performance lift is particularly significant for `ACORN` and `ACORN-γ`.
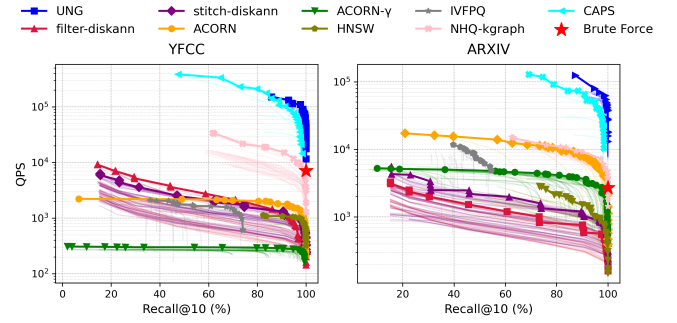
5.5 Effect of Varying Query Label Length

In all subsequent experiments, we use real-world label data. Similar to previous studies [3], we conduct experiments on two representative datasets for three scenarios. For each scenario, we categorize the size of the query label $|f_q|$ into three groups: short length, medium length, and long length. From each group, we randomly select 1,000 queries to explore the impact of query label length on the difficulty of the queries. Unless otherwise specified, all subsequent experiments are conducted on the two representative datasets, *arxiv* [30] and *yfcc* [46], and the three scenarios: *containment*, *equality*, and *overlap*. Since the original implementations and code of NHQ [21] and CAPS [15] do not support these scenarios, the subsequent experiments will not include comparisons with these two methods.

When the query labels are short, nearly all algorithms can achieve a high recall range (97%+) on both datasets. As shown in Figure 11, as the query label length increases, significant performance divergence in terms of recall emerges among the algorithms, accompanied by a general decrease in QPS. In the `Containment` and `Equality` scenarios, `UNG` exhibits superior performance due to its specialized design. Counter-intuitively, its QPS tends to increase as the labels get longer in these specific cases. A pre-filter brute force approach also performs remarkably well in these scenarios; although it employs a full-precision search, the number of candidate
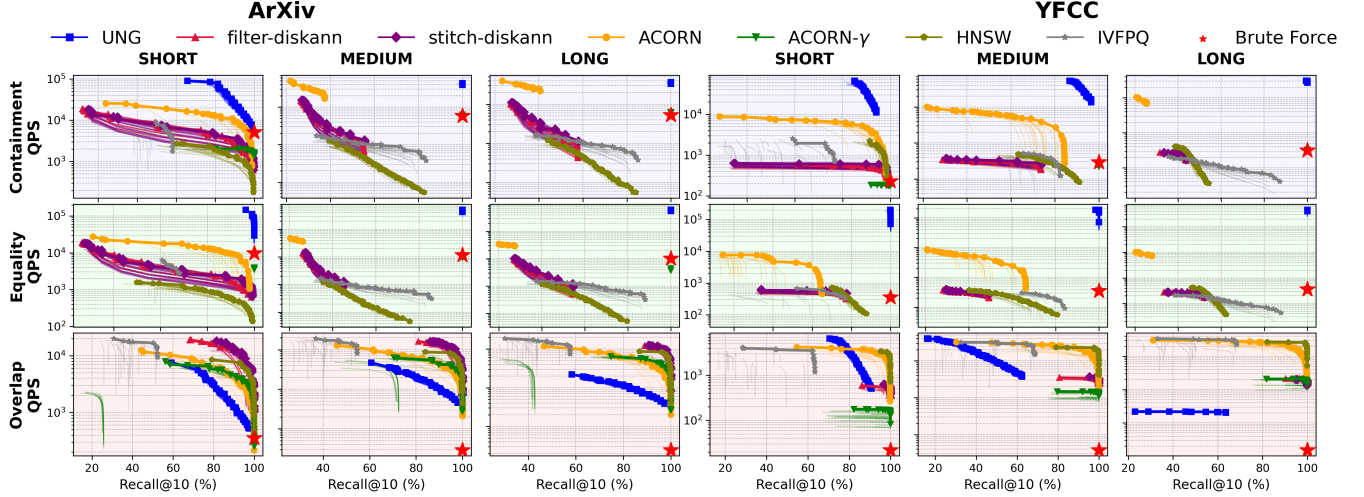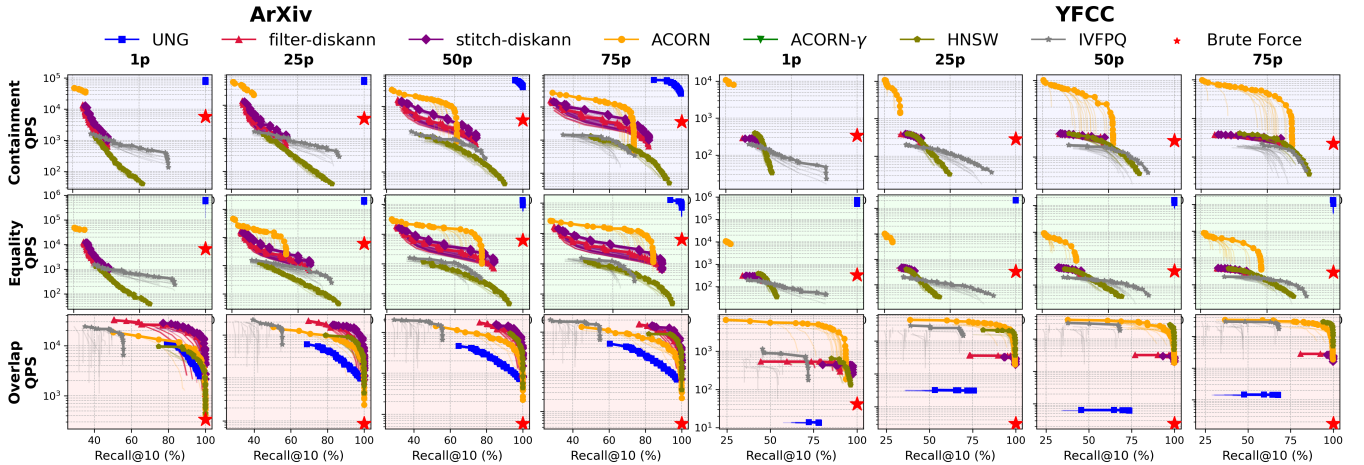
Figure 11: Results of Varying Query Label Length.

Figure 12: Results of Varying Query Selectivity.

points after filtering is small, thus limiting the number of distance computations and maintaining a relatively high QPS. Similarly, the performance of ACORN-γ often resembles the brute-force method, which can be attributed to cases where low selectivity effectively prunes the search space to a small number of candidates. In this scenario, as query labels grow longer, ACORN-1 and DiskANN algorithms quickly hit a bottleneck and struggle to improve recall. This is because during their search queue expansion, if the points in their neighbors do not satisfy the label constraints, it becomes difficult to expand to more distant points that do meet those constraints.

In contrast, the performance trends are reversed in the Overlap scenario. Here, methods such as filter-diskann, stitch-diskann, HNSW, and the ACORN variants all have the potential to be optimal under certain conditions. Notably, in this scenario, both the QPS and recall of UNG degrade rapidly as the label length increases.

5.6 Effect of Varying Query Selectivity

Similar to the setup in previous works [3, 14], we group the real-world query labels based on their selectivity $\sigma_S(f_q)$. According to the specific scenario, the queries are divided into four groups: 75th,

50th, 25th, and 1st percentiles. For each group, we evaluate the performance of different algorithms under each scenario.

In Figure 12, as selectivity increases from left to right, most methods exhibit an increase in recall while maintaining their QPS. UNG and ACORN typically retain higher QPS over a broad recall range, whereas post-filtering algorithms tend to lose throughput more rapidly at low selectivity. In containment and oequality scenario, UNG requires searching only a very small subgraph, so it does not encounter nodes that violate label constraints; consequently, lower selectivity yields higher QPS and recall approaches 100%.

In the overlap scenario, both DiskANN and ACORN achieve strong results because their neighbor selection retains only vectors with non-empty intersection with the query, which preserves recall. UNG shows no notable degradation on smaller datasets, but as the dataset grows it must re-rank many more candidates, causing both QPS and recall to drop.

5.7 Effect of Varying Top-k

Many real-world scenarios, such as face recognition and retrieval-augmented generation (RAG), have special requirements for both
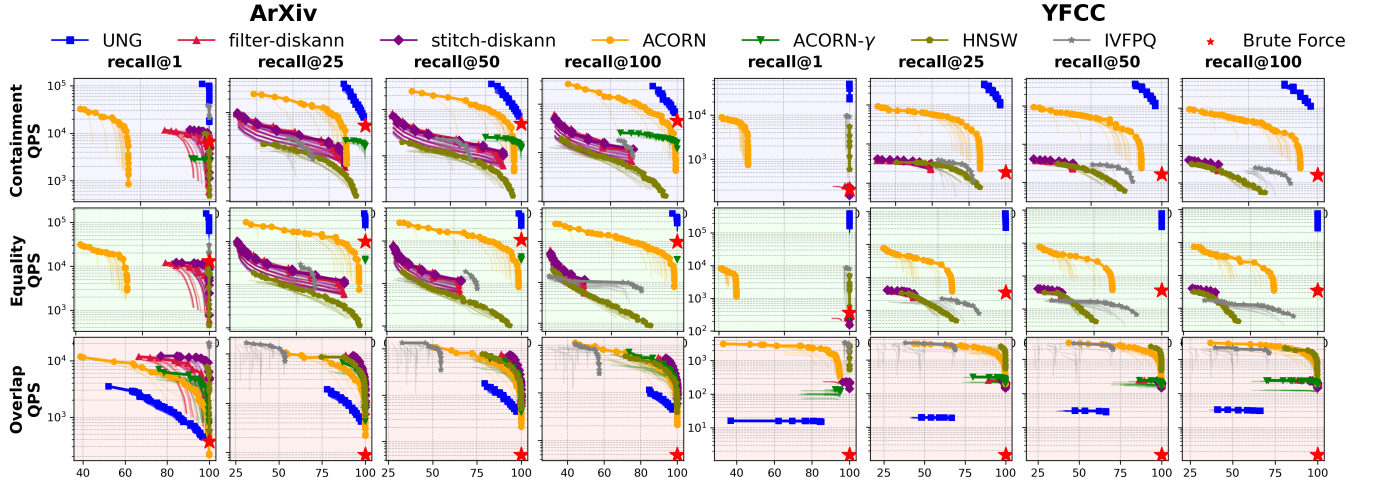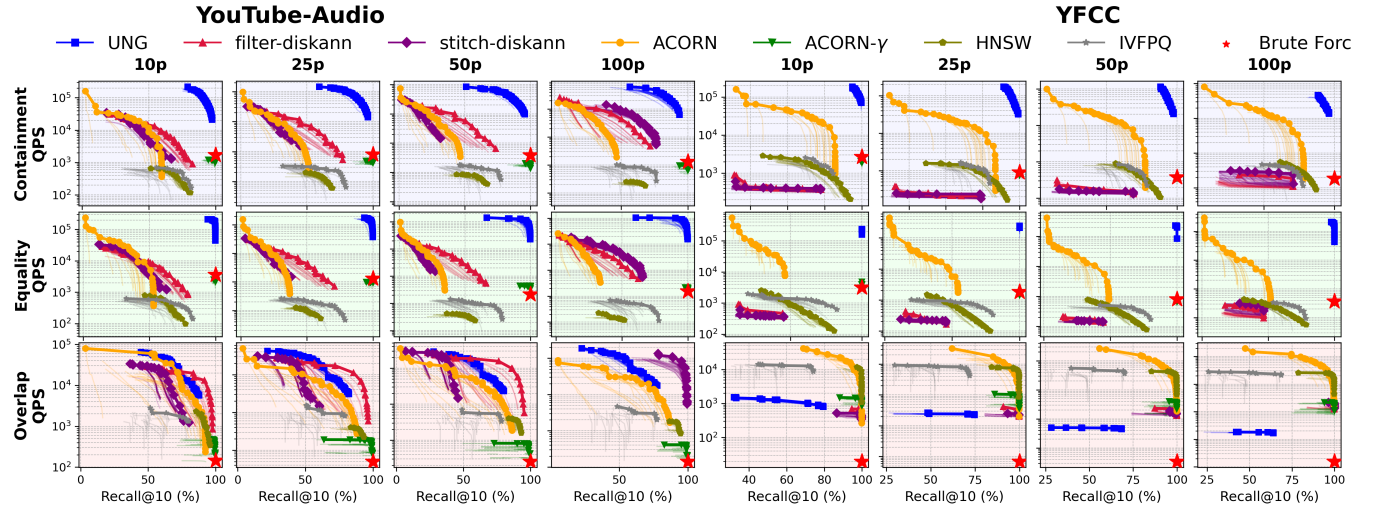
Figure 13: Results of Varying Top-k.

Figure 14: Results of Varying Base Dataset Size.

the selection of top-k and its accuracy [12]. Thus, in this set of experiments, we evaluate the impact of varying the value of *top-k* on the performance of different algorithms. For each scenario, we test a range of top-k values, including $k = 1$, $k = 25$, $k = 50$, and $k = 100$. We guarantee that the selected queries always have at least k ground truth results, regardless of the value of k.

As shown in Figure 13, across both the arXiv and YFCC datasets, the UNG and ACORN algorithm consistently demonstrates superior performance in the vast majority of scenarios. For instance, in the arXiv Containment scenario at recall@100, UNG reaches a QPS of 10^4 while ACORN sustains a QPS of approximately $5 * 10^3$ while achieving over 95% recall, whereas competing methods like stitch-diskann achieve lower QPS under the same conditions. noted that while post-filtering IVFPQ achieves high QPS and recall at recall@1 across all scenarios, its performance deteriorates significantly as the value of k in top-k increases. This is because the ground truth vectors can be scattered across different inverted buckets, making it very difficult to retrieve the complete set.

Similarly, the UNG algorithm demonstrates particularly poor performance in the Overlap scenario. In this context, while a considerable number of algorithms achieve comparable results on the arXiv dataset, their performance diverges significantly on the YFCC dataset, which features a larger number of labels. On the more challenging YFCC dataset, both ACORN and Post-filter HNSW stand out by maintaining exceptionally high and stable QPS and recall rates. Notably, their performance remains robust and does not degrade significantly even as the value of k increases.

5.8 Effect of Varying Base Dataset Size

To simulate the performance of different algorithms under varying base dataset sizes, we used two relatively large datasets: *yfcc* [46] and *youtube-audio* [1]. For each dataset, we selected subsets containing 10%, 25%, 50%, and 100% of the original data points. These subsets were then tested across three scenarios. We aim to analyze how the scalability of the algorithms is affected as the dataset grows.

In contrast to our earlier findings, increasing the dataset scale from 10% to 100% has a limited impact on performance. In Figure 14, the majority of algorithms exhibit only a marginal decrease in QPS, while their recall ranges remain largely stable. Consequently, the QPS-recall curves for a given scenario show minimal variation across different scales, indicating that dataset size is not the primary performance bottleneck. Inherently difficult queries that fail on the smaller dataset continue to do so on the larger one.

The choice of dataset reveals more pronounced differences. On YouTube-Audio, UNG and DiskANN are the top performers. Conversely, on the YFCC dataset, which features more intricate label relationships, the performance landscape is more varied. In these scenarios, UNG, ACORN, and HNSW each demonstrate distinct advantages. We must note that this entire evaluation is based on in-memory indexes; disk-based counterparts were not investigated.

Figure 15 illustrates the impact of dataset size scaling on indexing time across the YouTube-Audio and YFCC datasets. While the indexing time of most algorithms increases linearly as the dataset size grows, there are notable differences between the two datasets. Specifically, algorithms including ACORN-1, HNSW, and IVFPQ exhibit similar and relatively short indexing times, enabling efficient index construction—additionally, they do not require considering label distribution during the indexing process. In contrast, the remaining algorithms share comparable indexing times that are approximately one order of magnitude higher. Notably, these algorithms generally need to account for label distribution when building the index.

6 RECOMMENDATION

Method Evaluation We recommend evaluating methods primarily via QPS–Recall trade-offs and reporting Pareto-frontier points under consistent parameter sweeps. Assess stability across stratified query difficulties, including (i) label-length groups, (ii) selectivity percentiles, and (iii) varying top-k; note that varying the base dataset size typically has limited impact on search performance. Conduct experiments on representative datasets with diverse label distributions and cover comprehensive scenarios (*containment*, *equality*, *overlap*). Where appropriate, include index build time and memory footprint to decouple retrieval latency from construction cost, and report reproducible best-parameter sets for each method. Because many real-world datasets exhibit very low selectivity, it is crucial to emphasize evaluation under low-selectivity conditions, where throughput and recall often diverge most across methods.
Method Selection

- Prefer UNG as the default for strong QPS–recall in Containment and Equality scenarios; under very low selectivity or long labels, filter-then-search (e.g., UNG, ACORN-1, or even pre-filter Brute Force) keeps candidates small and throughput high; at high k, favor UNG/ACORN and avoid post-filter IVFPQ.
- Fixed-Length Equality is an easier, structured case where most methods reach high recall with proper tuning; UNG and CAPS offer top QPS at high recall.
- Prefer ACORN-1 and stitch-diskann in Overlap scenario for balanced throughput and recall; ACORN and post-filter HNSW are robust for large label spaces or high k; avoid UNG as query becomes more complex causing QPS/recall degradation.
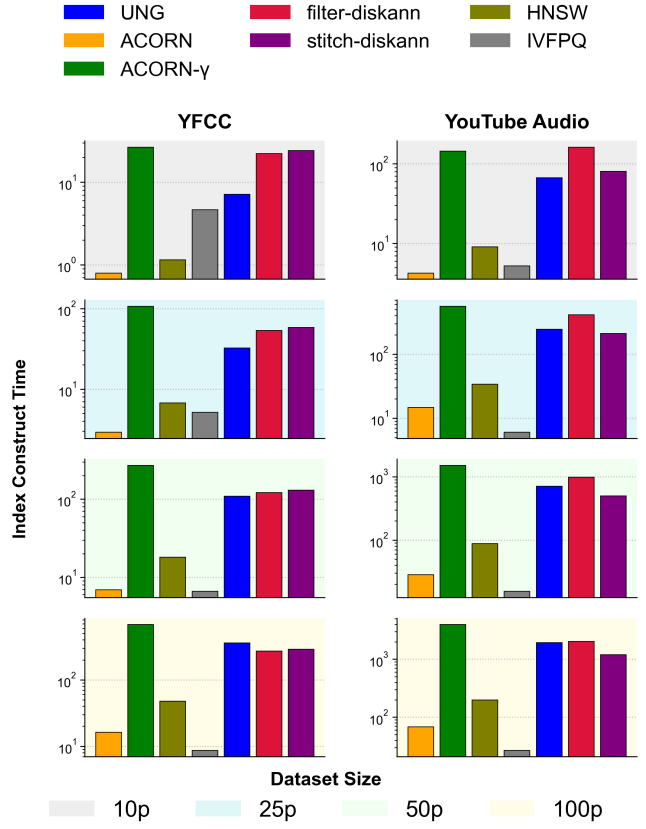

Figure 15: Effect of Varying Base Datasets Size on index construct time.

- At low k (e.g., recall@1), post-filter methods and UNG are competitive with high QPS; at higher k (e.g., recall@100), prefer UNG/ACORN for containment/equality and DiskANN/ACORN for overlap.
- If indexing time or memory is constrained, HNSW, IVFPQ, and ACORN-1 build quickly, are more memory-efficient, and offer greater flexibility as base labels change. Methods that rely on predicate label distributions (e.g., UNG, DiskANN variants) may incur higher build costs.

7 CONCLUSION

In this paper, we conducted a comprehensive benchmark and in-depth analysis of filtered Approximate Nearest Neighbor Search (FANNS). Our work addressed three obstacles: the combinatorial explosion of algorithmic configurations, the insufficient understanding of detailed impact factors, and the widespread fragmentation of evaluation methodologies. To overcome these challenges, we introduced a novel evaluation framework. By categorizing algorithms and employing a unified parameter tuning strategy, our framework mitigates evaluation bias and enables a fair, parameter-aware comparison that highlights the trade-offs of different algorithms. Through extensive experiments, we provided an in-depth analysis of how performance is affected by diverse factors, revealing the practical strengths and operational boundaries of each approach under realistic conditions.

REFERENCES

[1] 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. arXiv:1609.08675 [cs.CV] https://arxiv.org/abs/1609.08675

[2] Marián Boguñá, Dmitri Krioukov, and K. C. Claffy. 2008. Navigability of complex networks. *Nature Physics* 5, 1 (Nov. 2008), 74–80. https://doi.org/10.1038/nphys1130

[3] 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, Article 246 (Dec. 2024), 27 pages. https://doi.org/10.1145/3698822

[4] Benjamin Coleman, Santiago Segarra, Anshumali Shrivastava, and Alex Smola. 2021. Graph Reordering for Cache-Efficient Near Neighbor Search. arXiv:2104.03221 [cs.DS] https://arxiv.org/abs/2104.03221

[5] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web* (Hyderabad, India) (WWW '11). Association for Computing Machinery, New York, NY, USA, 577–586. https://doi.org/10.1145/1963405.1963487

[6] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The Faiss library. arXiv:2401.08281 [cs.LG] https://arxiv.org/abs/2401.08281

[7] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. New trends in high-d vector similarity search: al-driven, progressive, and distributed. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3198–3201.

[8] Elastic. [n.d.]. *Elasticsearch.* https://github.com/elastic/elasticsearch

[9] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2025. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. arXiv:1707.00143 [cs.LG] https://arxiv.org/abs/1707.00143

[10] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2024. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. arXiv:2409.09913 [cs.DB] https://arxiv.org/abs/2409.09913

[11] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 3, Article 167 (May 2024), 27 pages. https://doi.org/10.1145/3654970

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[13] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 4 (2014), 744–755. https://doi.org/10.1109/TPAMI.2013.240

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

[15] Gaurav Gupta. 2018. *constrainedANN.* https://github.com/gaurav16gupta/constrainedANN

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

[17] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management* (San Francisco, California, USA) (CIKM '13). Association for Computing Machinery, New York, NY, USA, 2333–2338. https://doi.org/10.1145/2505515.2505665

[18] Wenqi Jiang, Shigang Li, Yu Zhu, Johannes De Fine Licht, Zhenhao He, Runbin Shi, Cedric Renggli, Shuai Zhang, Theodoros Rekatsinas, Torsten Hoefler, and Gustavo Alonso. 2023. Co-design Hardware and Algorithm for Vector Search. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) (SC '23). Association for Computing Machinery, New York, NY, USA, Article 87, 15 pages. https://doi.org/10.1145/3581784.3607045

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

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

[21] KGLab-HDU. 2022. *TKDE-under-review-Native-Hybrid-Queries-via-ANNS.* https://github.com/KGLab-HDU/TKDE-under-review-Native-Hybrid-Queries-via-ANNS

[22] Jon Kleinberg. 2000. Kleinberg, J. Navigation in a small world. Nature 406, 845. *Nature* 406 (09 2000), 845. https://doi.org/10.1038/35022643

[23] Vihan Lakshman, ChoonHui Teo, Xiaowen Chu, Priyanka Nigam, Abhinandan Patni, Pooja Maknikar, and SVN Vishwanathan. [n.d.]. Embracing Structure in Data for Billion-Scale Semantic Product Search. ([n. d.]).

[24] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-Interest Network with Dynamic Routing for Recommendation at Tmall. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) (CIKM '19). Association for Computing Machinery, New York, NY, USA, 2615–2623. https://doi.org/10.1145/3357384.3357814

[25] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. 2021. Embedding-based Product Retrieval in Taobao Search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining.* https://doi.org/10.1145/3447548.3467101

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

[27] Anqi Liang, Pengcheng Zhang, Bin Yao, Zhongpu Chen, Yitong Song, and Guangxu Cheng. 2025. UNIFY: Unified Index for Range Filtered Approximate Nearest Neighbors Search. arXiv:2412.02448 [cs.DS] https://arxiv.org/abs/2412.02448

[28] Alessandro Magnani, Feng Liu, Suthee Chaidaroon, Sachin Yadav, Praveen Reddy Suram, Ajit Puthenputhussery, Sijie Chen, Min Xie, Anirudh Kashi, Tony Lee, and Ciya Liao. 2022. Semantic Retrieval at Walmart. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Washington DC, USA) (KDD '22). Association for Computing Machinery, New York, NY, USA, 3495–3503. https://doi.org/10.1145/3534678.3539164

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

[30] Malteos. 2022. Aspect Paper Embeddings. https://huggingface.co/datasets/malteos/aspect-paper-embeddings.

[31] Jiongkang Ni, Xiaoliang Xu, Yuxiang Wang, Can Li, Jiajie Yao, Shihai Xiao, and Xuecang Zhang. 2023. DiskANN++: Efficient Page-based Search over Isomorphic Mapped Graph Index using Query-sensitivity Entry Vertex. arXiv:2310.00402 [cs.IR] https://arxiv.org/abs/2310.00402

[32] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian (Allen) Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. 2019. Semantic Product Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* https://doi.org/10.1145/3292500.3330759

[33] Xichuan Niu, Bofang Li, Chenliang Li, Rong Xiao, Haochuan Sun, Honggang Wang, Hongbo Deng, and Zhenzhong Chen. 2020. Gated Heterogeneous Graph Representation Learning for Shop Search in E-commerce. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) (CIKM '20). Association for Computing Machinery, New York, NY, USA, 2165–2168. https://doi.org/10.1145/3340531.3412087

[34] OpenMP Architecture Review Board. 2008. OpenMP Application Program Interface Version 3.0. http://www.openmp.org/mp-documents/spec30.pdf

[35] Malte Ostendorff, Till Blume, Terry Ruas, Bela Gipp, and Georg Rehm. 2022. Specialized document embeddings for aspect-based similarity of research papers. In *Proceedings of the 22nd ACM/IEEE Joint Conference on Digital Libraries.* 1–12.

[36] Liana Patel. 2024. ACORN. https://github.com/stanford-futuredata/ACORN.

[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, Article 120 (May 2024), 27 pages. https://doi.org/10.1145/3654923

[38] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. *CoRR* abs/2103.00020 (2021). arXiv:2103.00020 https://arxiv.org/abs/2103.00020

[39] Navid Rekabsaz, Oleg Lesota, Markus Schedl, Jon Brassey, and Carsten Eickhoff. 2021. Tripclick: The Log Files of a Large Health Web Search Engine. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 2507–2513.

[40] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. 2021. LAION-400M: Open Dataset of Clip-filtered 400 Million Image-text Pairs. *arXiv preprint arXiv:2111.02114* (2021).

[41] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Dobson Manohar, Dmitry Baranchuk, Edo Liberty, Frank Liu, Ben Landrum, Mazin Karjikar, Laxman Dhulipala, Meng Chen, Yue Chen, Rui Ma, Kai Zhang, Yuzheng Cai, Jiayang Shi, Yizhuo Chen, Weiguo Zheng, Zihao Wan, Jie Yin, and Ben Huang. 2024. Results of the Big ANN: NeurIPS'23 competition. arXiv:2409.17424 [cs.IR] https://arxiv.org/abs/2409.17424

[42] Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, Andrija Antonijevic, Dax Pryce, David Kaczynski, Shane Williams, Siddarth Gollapudi, Varun Sivashankar, Neel Karia, Aditi Singh, Shikhar Jaiswal, Neelam Mahapatro, Philip Adams, Bryan Tower, and Yash Patel. [n.d.].

[43] Jitendra Nath Singh and Sanjay K. Dwivedi. 2015. Performance Evaluation of Search Engines Using Enhanced Vector Space Model. *Journal of Computer Science* 11, 4 (Jul 2015), 692–698. https://doi.org/10.3844/jcssp.2015.692.698

[44] SuhasJayaram Subramanya, Fnu Devvrit, HarshaVardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. *Neural Information Processing Systems,Neural Information Processing Systems* (Nov 2019).

[45] Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. 2019. *DiskANN: fast accurate billion-point nearest neighbor search on a single node.* Curran Associates Inc., Red Hook, NY, USA.

[46] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. 2016. YFCC100M: the new data in multimedia research. *Commun. ACM* 59, 2 (Jan. 2016), 64–73. https://doi.org/10.1145/2812802

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

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

[49] Wei Wu, Junlin He, Yu Qiao, Guoheng Fu, Li Liu, and Jin Yu. 2022. HQANN: Efficient and robust similarity search for hybrid queries with structured and unstructured constraints. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management.* 4580–4584.

[50] Tiannuo Yang, Wen Hu, Wangqi Peng, Yusen Li, Jianguo Li, Gang Wang, and Xiaoguang Liu. 2024. VDTuner: Automated Performance Tuning for Vector Data Management Systems. In *2024 IEEE 40th International Conference on Data Engineering (ICDE).* 4357–4369. https://doi.org/10.1109/ICDE60146.2024.00332

[51] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-Augmented Generation for AI-Generated Content: A Survey. *arXiv preprint arXiv:2402.19473* (2024).

[52] Chaoji Zuo and Dong Deng. 2023. ARKGraph: All-Range Approximate K-Nearest-Neighbor Graph. *Proc. VLDB Endow.* 16, 10 (June 2023), 2645–2658. https://doi.org/10.14778/3603581.3603601

[53] 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, Article 69 (March 2024), 26 pages. https://doi.org/10.1145/3639324