RWalks: Random Walks as Attribute Diffusers for Filtered Vector Search

ANAS AIT AOMAR, Mohammed VI Polytechnic University, Morocco
KARIMA ECHIHABI, Mohammed VI Polytechnic University, Morocco
MARCO ARNOBALDI, Oracle Labs, Switzerland
IOANNIS ALAGIANNIS, Oracle Labs, Switzerland
DAMIEN HILLOULIN, Oracle Labs, Switzerland
MANAL CHERKAOUI, Mohammed VI Polytechnic University, Morocco

Analytical tasks in various domains increasingly encode complex information as dense vector data (e.g., embeddings), often requiring filtered vector search (i.e., vector search with attribute filtering). This search is challenging due to the volume and dimensionality of the data, the number and variety of filters, and the difference in distribution and/or update frequency between vectors and filters. Besides, many real applications require answers in a few milliseconds with high recall on large collections. Graph-based methods are considered the best choice for such applications, despite a lack of theoretical guarantees on query accuracy. Existing solutions for filtered vector search are either: 1) ad-hoc, using existing techniques with no or minor modifications; or 2) hybrid, providing specialized indexing and/or search algorithms. We show that neither is satisfactory and propose RWalks, an index-agnostic graph-based filtered vector search method that efficiently supports both filtered and unfiltered vector search. We demonstrate its scalability and robustness against the state-of-the-art with an exhaustive experimental evaluation on four real datasets (up to 100 million vectors), using query workloads with filters of different types (unique/composite), and varied specificity (proportion of points that satisfy a filter). The results show that RWalks can perform filtered search up to 2x faster than the second-best competitor (ACORN), while building the index 76x faster and answering unfiltered search 13x faster.

CCS Concepts: • **Information systems → Information retrieval query processing**; **Database query processing**; **Nearest-neighbor search**.

Additional Key Words and Phrases: Vector search, Approximate nearest neighbor search, Hybrid search, Filtered search

ACM Reference Format:
Anas Ait Aomar, Karima Echihabi, Marco Arnobaldi, Ioannis Alagiannis, Damien Hilloulin, and Manal Cherkaoui. 2025. RWalks: Random Walks as Attribute Diffusers for Filtered Vector Search. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 212 (June 2025), 26 pages. https://doi.org/10.1145/3725349

1 Introduction

Motivation. Recently, there has been an unprecedented growth of large-scale vector data collections in various scientific and business domains, due in part to the proliferation of learned embeddings.

Authors' Contact Information: Anas Ait Aomar, Mohammed VI Polytechnic University, Benguerir, Morocco, anas.aitaomar@um6p.ma; Karima Echihabi, Mohammed VI Polytechnic University, Benguerir, Morocco, karima.echihabi@um6p.ma; Marco Arnobaldi, Oracle Labs, Zurich, Switzerland, marco.arnaboldi@oracle.com; Ioannis Alagiannis, Oracle Labs, Zurich, Switzerland, ioannis.alagiannis@oracle.com; Damien Hilloulin, Oracle Labs, Zurich, Switzerland, damien.hilloulin@oracle.com; Manal Cherkaoui, Mohammed VI Polytechnic University, Benguerir, Morocco, manal.cherkaoui@um6p.ma.

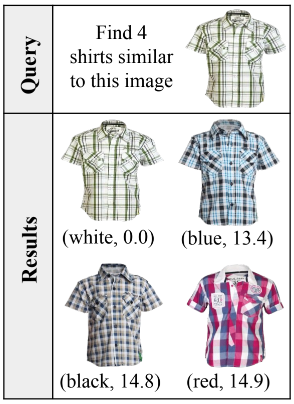
ACM 2836-6573/2025/6-ART212
https://doi.org/10.1145/3725349

These embeddings are dense vector representations learned by AI models to represent a variety of high-dimensional objects including text, images, video, graphs, tables, and code snippets.

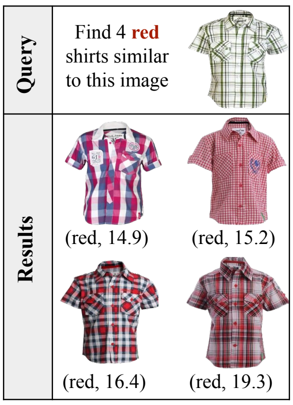
Analyzing this data at scale requires efficient and accurate vector search. The vector search problem has been studied extensively for over four decades, and has recently become a hot topic with the growth of analytical pipelines that require it as a core subroutine, including data discovery, classification, clustering, outlier detection, recommendation, and information retrieval [19, 22].

There exist different variations for the vector search problem. It can be filtered, i.e. vector search combined with attribute filtering (a.k.a. constrained or hybrid vector search) or unfiltered. The filters could include equality, inequality, set membership, logical (e.g. AND, OR) or range expressions. A popular use case of vector search is information retrieval in an e-commerce setting where customers can provide a query image and search for products that are similar to it. Oftentimes, additional filters are expressed such as a color, a price budget, or stock availability (Fig. 1).

Challenges. Vector search is challenging because the datasets can contain millions of objects, the dimensionality can reach thousands of dimensions, and users have demanding requirements in terms of the throughput and accuracy of analytical queries, often requiring answers on large-scale collections in a few milliseconds with high-recall. Filtered vector search is further challenging due to the number and variety of filters, and the fact that vectors and filters often do not have the same distribution and/or update frequency.



(a) Unfiltered query



(b) Filtered query

Fig. 1. Vector search in an e-commerce application. In (a), the user can only look for shirts similar to a query. In (b), the user can also specify the preference for red shirts.

The research community has proposed many vector search approaches, particularly for unfiltered vector search. Some are exact, while others are with or without query accuracy guarantees. We refer to the former as δ-ϵ-approximate and the latter as ng-approximate [23]. Some were designed for special data types such as data series [17, 20, 21, 52, 73] and others for generic high-dimensional vectors [10, 27, 59]. Recently, filtered vector search has been attracting more attention due to its importance for many AI applications [8, 41, 43, 44]. The proposed solutions are either: 1) *ad-hoc*, using existing techniques with no or minor modifications; or 2) *hybrid*, providing specialized indexing and/or search algorithms. We will demonstrate that existing solutions are unsatisfactory across diverse query workloads. For instance, the Filter-Then-Search strategy works well when filters are restrictive thus few vectors are kept to be ranked against the query vector, typically, via a sequential scan. However it cannot reach a high throughput when filters are more relaxed requiring to scan more candidates in the vector search phase. Search-Then-Filter exhibits trends opposite to Filter-Then-Search. It can lead to a good accuracy-efficiency trade-off when filters are relaxed but fail to guarantee acceptable recall when filters are restrictive. The specialized approaches for filtered search [71, 77] typically use attribute information when constructing the index. Therefore, they are impractical because the costly operation of building/updating the index should be performed every time attributes are updated, which happens more frequently than changes to the vectors themselves (e.g., a vector embedding for a product changes less frequently than an attribute representing its stock availability).

The `RWalks` Approach. In this paper, we propose RWalks, a technique for graph-based ng-approximate filtered vector search. Its key design features include: (1) a hybrid distance metric, (2) an index-agnostic attribute enrichment process, and (3) a distance pruning heuristic. RWalks operates independently of the underlying graph index structure, allowing it to support filtered search without modifying the index itself, thus preserving performance in unfiltered scenarios. It employs a hybrid distance metric that balances proximity to the query with filter satisfaction, guiding the search toward valid regions of the graph. To address the sparsity issue in low-specificity queries, where valid nodes are rare and dispersed throughout the graph, RWalks also implements a post-indexing attribute diffusion process which uses random walks to aggregate multi-hop information, enriching nodes with attributes from their surroundings and overcoming the limitations of local beam search when attributes are sparse. As a byproduct of enrichment, RWalks leverages the summarized multi-hop information as a heuristic for distance pruning during query answering. This reduces distance computations without affecting performance, as these pruned neighbors typically guide the search to regions unlikely to satisfy the query filters. Combining these features leads to consistent performance across all filter regimes without negatively impacting the core task of graph vector indexes, i.e., unfiltered search.

We make the following contributions:

- We present a survey of state-of-the-art filtered vector search methods and we perform a thorough analysis of their strengths and limitations according to key criteria, including query efficiency-accuracy trade-offs, indexing time, indexing footprint, and index-filter friction.
- We identify three desirable features for scalable filtered vector search: i) the index should efficiently support both unfiltered and filtered vector search with minimal impact on index maintenance; ii) the filter specificity should be decoupled from the vector search accuracy/efficiency trade-offs to ensure robustness in varied specificity workloads; and iii) both indexing and search should scale with larger dataset size, higher dimensionality, and different filter types (single/composite predicates).
- We propose RWalks, a novel index-agnostic[1] technique for graph-based ng-approximate vector search. To the best of our knowledge, RWalks is the first method to propose attribute enrichment for vector search, and exploit a hybrid distance during search on an index built in the vector space.
- We perform an ablation study that explains the contribution of individual design choices to the final performance across different dataset sizes and query specificities.
- We demonstrate the scalability, and robustness of RWalks against state-of-the-art approaches with an exhaustive experimental evaluation on five real datasets reaching up to 100 million vectors, using diverse query workloads with filters of different types (a single equality filter, and a composite "AND" filter) and varied specificity. The results show that RWalks can perform filtered search up to 2x faster than the second-best competitor (ACORN) while building the index 76x faster and answering unfiltered search 13x faster.

[1]This assumes the graph structure is designed for unfiltered vector search.

Scope. This paper proposes an index-agnostic framework for graph-based ng-approximate filtered vector search using the Euclidean distance. Support for other distance measures, types of search (e.g., range queries) and theoretical guarantees on query accuracy are out-of-scope and part of our future work.

2 Preliminaries

2.1 Definitions

The *ng*-approximate similarity search problem is typically abstracted as an *ng*-approximate k-NN search problem, where the (dis)similarity between objects is measured using a distance. The objects can be in metric or non-metric spaces, and in the latter, the distance fulfills symmetry, non-negativity and triangular inequality [11].

We consider a dataset $\mathbb{S} = \{V_{C_1}, V_{C_2}, \ldots, V_{C_n}\}$ consisting of n objects in a d-dimensional space. Each object V_{C_i} (i is the object identifier | $1 \leqslant i \leqslant n$) is associated with a p-dimensional **attribute vector** $A_{C_i} \in \{0,1\}^p$.

Definition 1. *An **attribute vector** $A_{C_i} \in \{0,1\}^p$ is a one-hot vector, where the presence or absence of an attribute for object V_{C_i} is denoted with a value of* 1 *or* 0 *respectively. The attribute vector of a query object V_Q is denoted as $A_Q \in \{0,1\}^p$.*

Definition 2. *A **filter** is a function $\mathbb{F} : \{0,1\}^p \times \{0,1\}^p \rightarrow \{0,1\}$ that checks if a data point's attribute vector A_{C_i} satisfies a query's attribute vector A_Q. Common filters include (j is the attribute position | $1 \leqslant j \leqslant p$) :*

- ***Equality**:* $\mathbb{F}_{eq}(A_{C_i}, A_Q) = 1 \iff \forall j,\ (A_Q[j] = 1 \implies A_{C_i}[j] = 1)$.
- ***AND**:* $\mathbb{F}_{and}(A_{C_i}, A_Q) = 1 \iff \forall j,\ (A_Q[j] = 1 \implies A_{C_i}[j] = 1)$.
- ***OR**:* $\mathbb{F}_{or}(A_{C_i}, A_Q) = 1 \iff \exists j,\ (A_Q[j] = 1 \land A_{C_i}[j] = 1)$.
- ***NOT**:* $\mathbb{F}_{not}(A_{C_i}, A_Q) = 1 \iff \forall j,\ (A_Q[j] = 1 \implies A_{C_i}[j] = 0)$.

Definition 3. *An ng-**approximate** k**-NN query** for a query object V_Q retrieves $\mathbb{A} = \{V_{C_1}, \ldots, V_{C_k}\} \subseteq \mathbb{S}$, a set of k approximate neighbors to V_Q, using heuristics without providing theoretical guarantees on retrieval quality.*

Definition 4. *A **filtered** ng-**approximate** k**-NN query** for a query object V_Q and a filter $\mathbb{F}$ retrieves the set of k ng-approximate neighbors $\mathbb{A} = \{V_{C_1}, \ldots, V_{C_k}\} \subseteq \mathbb{T}$, where $\mathbb{T} = \{V_{C_i} \in \mathbb{S} : \mathbb{F}(A_{C_i}, A_Q) = 1\}$. Here, $\mathbb{T}$ represents the **valid points** that satisfy the filter $\mathbb{F}$.*

Definition 5. *The **specificity** of a filtered ng-approximate k-NN query is the proportion of points in the dataset $\mathbb{S}$ that satisfy the filter $\mathbb{F}$. Formally: Specificity* $= |\mathbb{T}|/|\mathbb{S}|$

In the rest of the paper, we consider that objects are vectors in a d-dimensional space, and that the (dis)similarity between them is measured using the Euclidean distance.

2.2 Unfiltered Vector Search

2.2.1 Overview. The research community has proposed many techniques for the different flavors of unfiltered vector search. Although recent methods have improved the performance of exact [18, 23, 27] and δ-ϵ-approximate search [33, 35], they are still considered too slow for many AI applications [40, 61, 64, 80]. Therefore, ng-approximate search attracted increasing attention in the past decade [24, 45]. Different data structures were proposed to support ng-approximate vector search including scans [27, 75], trees [2, 24, 50], graphs [28, 30, 47, 51, 65, 70], inverted indexes [6, 37, 38, 78], or a mix of different structures [4, 9, 28, 39, 51, 81]. Graph-based techniques are considered nowadays the vector search method of choice for AI applications that can sacrifice theoretical guarantees for empirical performance [5, 40, 61, 64, 80].

2.2.2 Graph-Based Methods. Graph-based vector search approaches typically build a proximity graph [31, 60, 69], where nodes represent vectors in the dataset, and edges connect similar vectors. State-of-the-art graph-based methods such as Vamana [65], HNSW [47], NSG [30], Elpis [4], and others [9, 15, 28, 29, 39, 45, 46, 51] differ in how they build the graph, but they all use the same beam search algorithm [56]. Search starts at an entry node (which can be chosen randomly or using a more sophisticated approach) then visits neighboring nodes in a best-first, greedy manner, stopping when no better candidates are found [56]. Below, we briefly describe some key graph-based methods.

HNSW [47] builds multiple NSW graphs [46, 54] with RNG diversification [68] and organizes them into a hierarchical structure, where each layer includes all nodes in the layer above it, with the bottom (a.k.a. *base*) layer containing all points of the dataset. An NSW graph is an approximation of a Delaunay graph which guarantees the small world property [42, 74], i.e., the distance between any two randomly chosen nodes grows proportionally to the logarithm of the dataset size. During search, HNSW uses the hierarchical layers to efficiently locate an entry point in the base layer to start search.

NSG [30] builds an approximate k-NN graph, applies RNG diversification [68], and adds bidirectional edges between nodes and their neighbors. To ensure the connectivity of the graph, NSG constructs a tree from a depth-first search graph traversal and adds an edge between any disconnected node and its nearest node in this tree.

Vamana [65] first builds a randomly generated graph with a node degree greater than the logarithm of the dataset size, to ensure graph connectivity [25]. Then, it uses a relaxed RNG process to diversify the graph, and adds bi-directional edges. An RNG process is applied on nodes that exceed the maximum out-degree, and an additional relaxed RNG pass takes place to improve the graph connectivity.

Elpis [4] is a divide-and-conquer approach, that partitions a dataset into multiple clusters using the Hercules EAPCA tree [18], where each leaf corresponds to a different cluster. Then, Elpis builds in parallel an HNSW index for each leaf. During search, Elpis leverages the EAPCA summarization and lower-bounding distance [73] to prune clusters, then runs multiple concurrent beam search algorithms on non-pruned clusters.

2.3 Filtered Vector Search

Several works have been proposed recently for filtered vector search, i.e., vector search with additional conditions such as filtering by specific attributes, categories, or ranges. We identify two different classes: *ad-hoc* methods which use existing techniques with no or minor modifications; and *hybrid* methods that provide specialized indexing and/or search algorithms. Below, we describe the recent state-of-the-art methods in each class, highlighting their strengths and limitations.

2.3.1 Ad-hoc methods. These methods use existing unfiltered vector search methods with minimal or no modifications.

Filter-Then-Search (FTS) first applies the filtering criteria to reduce the number of candidates. This can be done using an existing index on top of the attributes, e.g., a B-tree [13]. It then executes vector search, using a sequential scan since the returned list is not known in advance. This strategy guarantees a 100% recall if the filtering index is exact and a sequential scan is used to refine the vectors. It is effective when the filters have low specificity. However, when filters have high specificity, the sequential scan has to process a large number of candidates, thus leading to high query latency.

Search-Then-Filter (STF) first retrieves a set of k' candidates using an unfiltered vector search approach, where k' is chosen to be higher than the requested number of neighbors k. Then, it refines this set to retain only vectors that satisfy the specified filters. This strategy is often faster than FTS since it leverages state-of-the-art unfiltered vector search indexes. However, it typically suffers from low recall in low-specificity workloads. This is because the filtering step may return less than the k answers requested by the user, and an appropriate value for k' may not be known a-priori.
Inline Processing with Graph-Based Indexes modifies a graph-based vector search algorithm, such as HNSW, by conditioning the addition of points to the result set. This approach does not require post-search filtering. However, as we will explain in more detail in § 3.1, this approach can increase search latency by correlating the length of graph traversal with the specificity of the filters.
Inline Processing with Inverted Indexes [57] modifies the search algorithm of an inverted index by scanning only points that meet specific query filters. This reduces the number of distance calculations, especially in low-specificity regimes. However, as specificity decreases, more clusters (posting lists) need to be scanned because points meeting the query filters are less common. Additionally, the inverted index search complexity scales linearly with data size. Therefore, on large datasets, it should be combined with other data structures, such as HNSW, to represent each posting list.

2.3.2 Hybrid Methods. Recently, novel hybrid methods have been proposed to support filtered vector search by integrating filtering information into vector search or index construction. Below, we describe three state-of-the-art hybrid methods.
HQANN [77] modifies the HNSW index construction algorithm to use a hybrid distance, that considers the distances in the vector and attribute spaces, often prioritizing the latter. This strategy organizes points into clusters within the graph sharing similar attributes. HQANN is effective in scenarios with low specificity where clusters are small. Thus, a short traversal is enough to capture all valid answers. However, in high-specificity settings, points that are similar in vector distance may be located far from each other due to the focus on attribute distance resulting in larger-size attribute-based clusters. This can lead to increased latency during the search.
Filtered-DiskANN [32] extends the graph-based vector method Vamana [36], by incorporating both vector geometric relationships and attribute information during graph construction. Rather than using a hybrid distance metric, it modifies the graph structure through an edge-pruning process that enforces the retention of edges between nodes sharing similar attributes. Filtered-DiskANN currently only supports queries with equality filters.
ACORN [53] proposes two variants to HNSW based on the idea of expanding the HNSW node neighborhoods: ACORN-G and ACORN-1. ACORN-G constructs a denser HNSW graph by increasing the number of neighbors proportional to the smallest filters specificity, then filters this expanded neighbor list during the search to evaluate only those satisfying the query filters. In contrast, ACORN-1 delays the neighbor expansion until the search phase, approximating ACORN-G's dense graph while reducing indexing time at the cost of higher search latency. A notable limitation of ACORN is the disabling of RNG pruning strategy [9, 30, 47], which is crucial for ensuring efficient performance in unfiltered search scenarios [47].

2.4 Summary

Table 1 compares the state-of-the-art filtered vector search approaches in terms of search (recall/latency) and indexing performance (footprint, time and index/attributes friction). Ad-hoc methods do not incur any indexing overhead because they use existing indexes without altering their construction algorithm. However, as discussed earlier, their search performance is unstable across query workloads with varied specificity. In low-specificity workloads, STF and HNSW-Inline suffer from low accuracy and high latency respectively. In high-specificity workloads, FTS and IVF-Inline both deteriorate in efficiency. In contrast, existing hybrid methods HQANN and Filtered-DiskANN adapt the index to support filtered search, optimizing for recall at the expense of efficiency. Besides, they introduce a dependency on attributes during index construction, which can negatively affect index maintenance and scalability. Although ACORN demonstrates strong search performance, its index construction is tightly linked to filters specificity, which can result in significantly increased indexing times.

The comparison of filtered vector search methods reveals three key desirable features for efficiency and robustness: 1) the filter specificity should be decoupled from the recall-latency trade-off to ensure the method is robust in query workloads of varied specificity; 2) the index should be constructed to support both filtered and unfiltered vector search with minimal impact on index maintenance (e.g. to avoid frequent graph updates and/or costly re-builds); 3) both index construction and search should scale well with large datasets, high dimensions, and different filtering types.

Table 1. Performance comparison

🔴 Poor 🟡 Fair 🟢 Good

		Search Performance		Index Performance			
		Recall	Latency	Scalability	Attribute Dep.	Memory	TTI
Ad-hoc	FTS	🟢	🔴	🟡	🟢	🟢	🟢
	STF [1]	🔴	🟢	🟢	🟢	🟢	🟢
	IVF-Inline [57]	🟢	🟡	🟡	🟢	🟢	🟢
	HNSW-Inline [1]	🟢	🔴	🟢	🟢	🟢	🟢
Hybrid	HQANN [77]	🟢	🟡	🟡	🔴	🟡	🟡
	Filtered-DiskANN [32]	🟢	🟡	🟢	🔴	🟢	🟡
	ACORN-G [53]	🟢	🟢	🟡	🟢	🟡	🔴
	ACORN-1 [53]	🟢	🟡	🟡	🟢	🟢	🟢
	RWalks	🟢	🟢	🟢	🟢	🟡	🟢

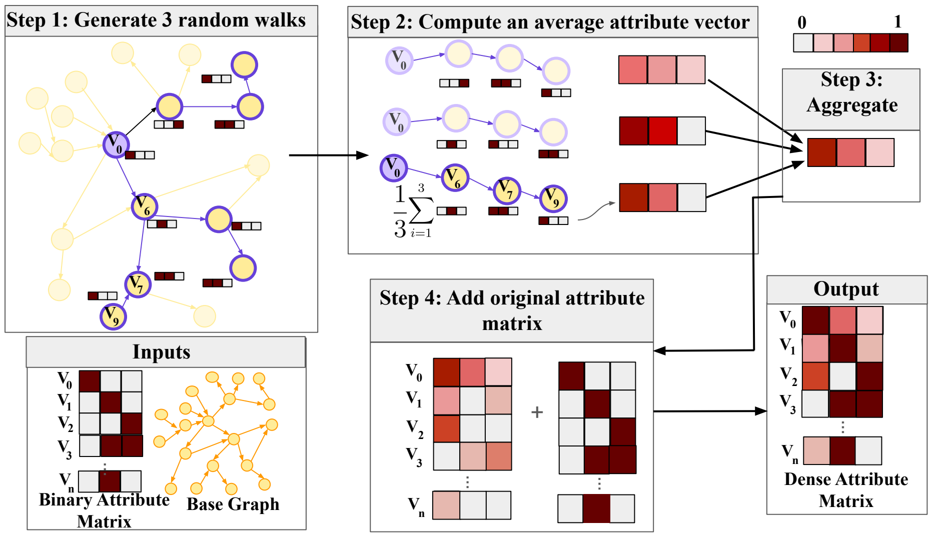
3 Proposed Approach: RWalks

We introduce RWalks, a novel hybrid vector search approach that boasts the three desirable features discussed earlier.

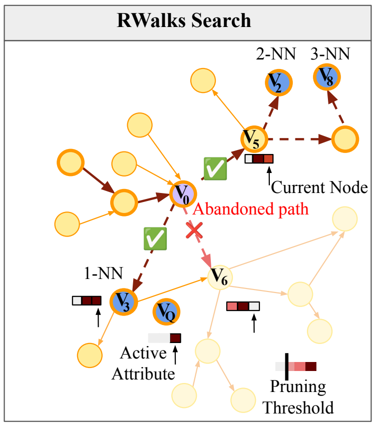
RWalks supports both filtered and unfiltered vector search thanks to: 1) a post-processing algorithm that can enrich any existing graph-based vector index with attribute information (without altering the original index), and 2) a query-answering algorithm that modifies the greedy search used by state-of-the-art approaches to support filtered vector search. RWalks is index-agnostic; therefore, it can be used with any graph-based vector search method. However, for the sake of clarity, and unless otherwise specified, we discuss RWalks in the context of HNSW since it is one the most popular baselines [3, 62].

3.1 Relaxing Specificity/Latency Dependency

Query answering using HNSW in the unfiltered case


Step 1: Generate 3 random walks
Step 2: Compute an average attribute vector
0
1
Step 3: Aggregate
$\frac{1}{3}\sum_{i=1}^{3}$
Step 4: Add original attribute matrix
Output
Inputs
Binary Attribute Matrix
Base Graph
Dense Attribute Matrix

(a) RWalks attribute enrichment workflow



(b) RWalks search with pruning

Fig. 2. RWalks workflow

consists of traversing a graph using a greedy search algorithm and populating a priority queue (W) with potential neighbors. The size of this queue acts as a termination condition, controlling the search latency. To support filtered search, HNSW-Inline modifies the criterion for adding candidates to W: instead of adding candidates based on their distance to the query only, it adds them only if they satisfy the query filters. This ensures recall, as all points aggregated in W satisfy the query filters. However, it can also slow down the graph traversal in low-specificity workloads since filling up W takes a longer time.

Removing this controlled addition of candidates to W results in stable traversal length regardless of specificity, but may affect recall guarantees as not all points in W validate the filters. That is why solving for latency affects recall. To alleviate this issue, we build upon the following observation:

Observation 1. Given the bounded size of W, points in the queue are popped when better candidates are found (i.e., closer to the query). Yet, these popped elements may satisfy the query filters and should not be discarded.

We introduce a twin priority queue (W_t) where popped elements that meet the query filters are added. Once the search is complete, W_t is merged with W. This simple modification reduces the drop in recall compared to a baseline HNSW search while stabilizing latency, as we eliminate the controlled addition of points to W thus keeping the traversal length the same as in an unfiltered search.

3.2 Improving Recall with a Biased Traversal

The proposed search modification based on **Observation 1** maintains stable latency across various filter regimes, shifting the impact of filter specificity from latency to recall, albeit not to the same extent as a baseline HNSW search. For instance, with a uniform distribution of attributes (e.g., the publication year of a paper not correlating with its topic), the ratio of valid points (those meeting the query filters) encountered during graph traversal will mirror the filter specificity. Therefore, lower specificity leads to fewer valid points, resulting in fewer candidate neighbors and consequently, lower recall. This effect is amplified if the attribute distribution negatively correlates with the vectors (e.g., searching for papers similar to RWalks but tagged under the astronomy field).

We address this limitation by proposing a biased graph traversal that prioritizes visiting valid points. We achieve this by using a hybrid distance metric during search, unlike HQANN that uses a hybrid distance during both index construction and search (which is not optimal as we will demonstrate in § 5).

3.3 Relaxing Sparsity with Graph Diffusion

In low-specificity regimes, particularly if there is no clustering

(points in $\mathbb{T}$ are uniformly distributed in the graph), this hybrid distance may become ineffective. As explained in our second observation below.

Observation 2. In low-specificity workloads with no clustering, few nodes share the same attributes as the query, i.e., the probability of a traversed node being valid is low (attribute sparsity), rendering obsolete the attribute component of the hybrid distance.

During graph traversal, neighbors of visited nodes are evaluated using a binary question: does this neighbor satisfy query filters? We enrich the nodes' attribute representation to answer the additional question: would this neighbor guide us toward nodes that satisfy query filters? We achieve this with a novel post-indexing attribute enrichment process (Alg. 1) that transforms the attribute representation from a binary form that encodes only the attributes of a given node, to a dense representation that is aware of the attributes of a node and its surroundings.

We exploit graph diffusion, inspired by label propagation [34] in node classification where labeled nodes propagate their labels to their unlabeled neighbors.

D-hop Random Walks as Attribute Diffusers. RWalks creates a rich attribute representation for each node V_i ($1 <= i <= n$; each vertex V_i represents a point V_{C_i}), by running multiple random walks starting from V_i, on an existing

graph, and aggregates the attributes of all visited nodes via an average. Fig. 2a illustrates this workflow on node V_0.

How to Construct Rich Attribute Representations? Let V_i and $h_{V_i} \in \{0,1\}^p$ denote the processed node and its binary attribute vector, respectively, where p is the cardinality of the attribute set. Generating rich attribute representations (Alg. 1) for V_i involves m random walks, each of length D, represented as RW_j for $j = 1, 2, \ldots, m$ (line 3).

Each walk rooted at node V_i visits D nodes, denoted as $\{V_{i,1}, V_{i,2}, \dots, V_{i,D}\}$. Here, $V_{i,1} = V_i$ denotes the root node, and $V_{i,j}$ for $j = 2, \dots, D$ are the subsequent nodes visited by that walk. For instance Step 1 in Fig. 2a shows the generation of ($m = 3$) random walks rooted at node V_0.

Now, we can create a rich attribute vector (lines 7–11), denoted as $AVG(RW_j)$, for each random walk RW_j by averaging the attribute vectors of the observed nodes in the walk. This process is described in Step 2 in Fig. 2a. The final dense attribute vector $E_{V_i} \in \mathbb{R}^p$ for the processed node V_i is obtained by averaging the contextual attribute vectors of all random walks (lines 12–13).

E_{V_i} will be utilized in the hybrid distance during the search process as described in Alg. 2.

Algorithm 1: RWalks-Indexing-Enrichment

```
   Input: node (node to be enriched), G (base graph), m (number of random walks), D (depth
          of random walks), p (attribute vector dimension)
   Output: E (enriched attribute vector)
1  Initialize an empty vector E of size (p);
2  for int i = 0 to m − 1 do
3      get a D-hops random walk walk starting at node;
4      Initialize a vector result of size (p);
5      for int k = 0 to D − 1 do
6          current_node ← walk[k];
7          node_attr ← getAttributeVector(current_node)
8          for i = 0 to p-1 do
9              result[i] ← result[i] + node_attr[i];
10     for int i = 0 to p − 1 do
11         result[i] ← result[i]/D;
12     E ← E + result (element-wise addition);
13 E ← E/m;
14 return E;
```

3.4 Pruning Distances with Attribute Heuristics

A key difference between graph-based indexes and other filtered vector search approaches is the timing of the distance calculations in the vector space. In serial scans and inverted indexes, the distance between the query and a candidate vector is evaluated only if the candidate satisfies the filters. However, in graph-based methods, the distance between the query and a visited node must be computed even if the latter does not satisfy the filters. This is because the node can be a candidate for future nodes to probe (access to new parts of the graph), allowing for continued graph traversal. In other words, during search, at any given node in the graph traversal, neighbors can be partitioned into two classes, either they validate the query filters or they do not. In low-specificity workloads, the set of valid neighbors may be small or even empty.

We described how the rich attribute representations in RWalks address attribute sparsity and improve query recall. Now, we will illustrate how these representations can also be leveraged to improve query efficiency. In RWalks, the neighbors of a visited node can be partitioned into three classes: 1) neighbors that satisfy the query filters; 2) neighbors that do not satisfy the query filters but might guide the traversal toward valid regions in the graph; and 3) neighbors that do not satisfy the query filters nor guide search toward valid regions. Fig. 2b illustrates these three classes of neighbors for node V_0: V_3 satisfies the query filter, V_5 is not a valid point but guides the traversal to valid points $\{V_2,V_8\}$, and V_6 is not a valid point and does not guide us to other valid points. To improve search efficiency, we propose the following heuristic: during search, only nodes that belong to the first two classes are visited. The choice of neighbors to be evaluated is done by z-normalizing the attribute vectors in the indexing process, then applying during search a user-defined threshold $\tau >= 0$ (zero is the mean) to decide which neighbors to visit. We z-normalize the attributes to be able to use the same parameter τ independently of the filter specificity (specificity and distribution affect the mean value for each set of attributes).

To illustrate the reduction in distance calculations, we compare the number of operations (distance calculations) at a specific hop during graph traversal. The initial operations count is given by: $d\_count = m_0 * (d + p)$ where d and p represent the dimension of data and attribute vectors respectively, and m_0 the number of neighbors. The number of distance calculations after applying τ becomes: $d\_count(\tau) = valid_{\text{ratio}}(\tau) * m_0 * (d + p)$. In this last equation, the term $valid_{\text{ratio}}(\tau)$ accounts for the ratio of neighbors evaluated. Note that using a higher threshold τ, thus leading to a smaller $valid_{\text{ratio}}$, can significantly reduce the number of distance computations. However, this may prematurely terminate the search and thus affect the quality of search (recall saturation).

3.5 The RWalks Search Algorithm

We now present Alg. 2, the RWalks search algorithm that exploits the ideas discussed in § 3.1, 3.2, 3.3 and 3.4, to support both filtered and unfiltered vector search. Note that most state-of-the-art graph-based vector search methods answer queries using the same beam search algorithm [72]. We highlight the modifications to this beam search with underlined text.

The search process involves iteratively extracting the closest node from the set of candidates C, evaluating its neighbors, and adding them to the set of results W if they are closer in distance to the query (line 13). Note that the algorithm computes two distances: $distV$, the distance in vector space, and $distH$, the hybrid distance. The set of candidate nodes is processed until it becomes empty (line 2), or a distance termination condition is reached (line 5). The distinctions from a baseline HNSW search are: 1) the additional priority queue W_t that keeps track of valid elements popped during the graph traversal (lines 18-19); 2) the hybrid distance metric $distV + h * d_{attr}$ that is used to bias the graph traversal (line 13); and 3) the distance pruning heuristic that prevents the exploration of unpromising paths (line 10). The function *checkPromisingElement* (line 11) depends on the filtering interface required. For a simple equality constraint (e.g., $e\_attr[\textit{index of query attribute}] > \tau$), we check if the evaluated node's attribute (after enrichment) exceeds the pruning threshold (τ). This can be generalized to an OR filter, and thus to a NOT filter, since the latter can be modeled as an OR filter on the remaining attributes, where at least one attribute must exceed the threshold. For an AND filter, The function *checkPromisingElement* (line 11) is a dot product between the attribute enriched vector and the query attribute vector divided by the number of active filters/attributes. This quantity is then compared to the pruning threshold (τ) defined by the user.

4 Theoretical Analysis

Framework. As detailed in § 3, improving recall in filtered search without modifying the base graph requires increasing the ratio of valid points encountered during traversal. RWalks achieves this by biasing the traversal with a hybrid distance and enriched attribute representations. We formally show how these modifications boost the ratio of valid points in the search. Let $P_{\text{obs}} = P \cup \left(\bigcup_{x_i \in P} N(x_i)\right)$ represent the observable nodes in the traversal, where P is the search path. To demonstrate the effectiveness of RWalks, we present propositions showing that R_i, the ratio of valid nodes in P_{obs} when using the hybrid distance and attribute enrichment, increases compared to using only the hybrid or vector distances. Consider $G = (V, E)$, a graph where each node $v \in V$

```
Algorithm 2: RWalks-Search
Input: query vector $q$, query attribute vector $A_Q$, entry point $ep$, results set size $ef$
Output: $W_t$
// $W$: results set; $W_t$: twin results set
// $C$: candidates set ; $V$: visited set
1  $V, C, W, W_t \leftarrow ep$ // Initialize all sets with ep
2  while $|C| > 0$ do
3      $c \leftarrow$ nearest element in $C$ to $q$;
4      $f \leftarrow$ furthest element in $W$ to $q$;
5      if $distV(c, q) > distV(f, q)$ then
6          break ;      // We terminate the search when the next best candidate is less
             similar than the furthest neighbor currently in the result set.
7      for each $e \in neighborhood(c)$ do
8          if $e \notin V$ then
9              $V \leftarrow V \cup \{e\}$ ;                    // mark point as visited
10             if checkPromisingElement($e_{attr}$) ;    // $e_{attr}$ denotes the attribute vector
                associated with element e
11             then
12                 $f \leftarrow$ furthest element in $W$ to $q$;
13                 if $distH(e, q, e_{attr}, A_Q) < distH(f, q, f_{attr}, A_Q)$ or $|W| < ef$ then
14                     $C \leftarrow C \cup \{e\}$;
15                     $W \leftarrow W \cup \{e\}$;
16                     if $|W| > ef$ then
17                         remove furthest element in $W$ to $q$;
18                         if popped element is valid then
19                             $W_t \leftarrow W_t \cup \{furthest\ element\}$;
   // Merge results sets
20 for each $e \in W$ do
21     if e is valid then
22         $W_t \leftarrow W_t \cup \{e\}$;
23     if $|W_t| > ef$ then
24         remove the furthest element from $W_t$ to $q$;
25 return $W_t$
```

has an attribute $a(v) \in \{0, 1\}$. Assume a valid query q, i.e. $a(q) = 1$. Our proofs focus on a single binary attribute for clarity, but they can be easily extended to multiple attributes.

Proposition. Let s and s' be the probabilities of selecting a valid node under the vector and hybrid distances respectively, $s' \geq s$.

Proof. Let $distV(v, q)$ and $distH(v, q) = distV(v, q) + h(1 - a(v)a(q)), h > 0$, be the vector and hybrid distances respectively. $distH(v, q) = distV(v, q)$ if $a(v) = 1$ and $distH(v, q) = distV(v, q) + h$ if $a(v) = 0$. Each node has M neighbors, with m_1 valid and $m_0 = M - m_1$ invalid. Let $X_1, \ldots, X_{m_1}$ be the distances to valid neighbors and $Y_1, \ldots, Y_{m_0}$ the distances to invalid neighbors. We assume that $distV(v, q)$ are i.i.d. with cumulative distribution function $F(t) = P(distV(v, q) \leq t)$.

For valid neighbors ($a(v) = 1$), $distH(v, q) = distV(v, q)$ implies $X_i \sim F(t)$. For invalid neighbors ($a(v) = 0$), $distH(v, q) = distV(v, q) + h$ implies $Y_j \sim G(t) = F(t-h)$. Given $F(t)$ is non-decreasing and $h > 0$, $G(t) = F(t-h) \leq F(t)$, making Y_j stochastically larger than X_i [76]. Consequently, $Y_{(1)} = \min\{Y_1, \dots, Y_{m_0}\}$ is stochastically larger than $X_{(1)} = \min\{X_1, \dots, X_{m_1}\}$.

The probability s' of selecting a valid node under the hybrid distance is $s' = P(X_{(1)} \leq Y_{(1)})$. Since $Y_{(1)}$ is stochastically larger that $X_{(1)}$, and given s as the probability under vector distance, we find: $s' \geq P((X_{(1)} \leq Y_{(1)} \mid \text{X}_{(1)}$ and $Y_{(1)}$are i.i.d following $F(t)) = s$, therefore, $s' \geq s$. □

Proposition. The ratio of valid nodes $R_2 \geq R_1$, where R_2 is under hybrid distance and R_1 under vector distance.

Proof. Consider the search path $P = \{x_0, \dots, x_p\}$ of length p, where each node x_i has M neighbors.

Under the vector distance, with the probability of selecting a valid node as s, the expected number of valid nodes in P is $p \cdot s$. Each node V_i has M neighbors, leading to the expected number of valid neighbors as $M \cdot s$. Therefore, the total expected valid nodes in P_{obs} is $E_{\text{tot}}^{(1)} = p \cdot s + p \cdot M \cdot s$. The size of P_{obs} is $|P_{\text{obs}}| = p + p \cdot M$, thus the valid node ratio under the original metric is $R_1 = \frac{E_{\text{tot}}^{(1)}}{|P_{\text{obs}}|} = s$.

Under the hybrid distance, with the probability of selecting a valid node as s', the expected number of valid nodes in P is $E_{\text{tot}}^{(2)} = p \cdot s' + p \cdot M \cdot s$. The valid node ratio under the hybrid metric then becomes $R_2 = \frac{E_{\text{tot}}^{(2)}}{|P_{\text{obs}}|} = \frac{s' + M \cdot s}{1+M} \geq \frac{s + M \cdot s}{1+M} = R_1$. □

Let s'' be the probability of selecting a valid node under the hybrid distance with enriched attributes, $s'' \geq s'$. .

Proof. The enriched distance is defined as

$$distH_{\text{enr}}(v, q) = distV(v, q) + h(1 - \tilde{a}(v)a(q)), \ \ h > 0,$$

where the enriched attribute $\tilde{a}(v)$ is: (i) $\tilde{a}(v) = 1$ if $a(v) = 1$; (ii) $\tilde{a}(v) = \alpha$ if $a(v) = 0$ and at least one valid node within D hops or $\tilde{a}(v) = 0$ if $a(v) = 0$ and all nodes within D hops are invalid; where $0 < \alpha < 1$.

Types of nodes based on the value of $\tilde{a}(v)$ are: 1) valid nodes $\tilde{a}(v) = 1$: $distH_{\text{enr}}(v, q) = d(v, q)$; 2) invalid nodes leading to valid regions $\tilde{a}(v) = \alpha$: $distH_{\text{enr}}(v, q) = d(v, q) + h(1-\alpha)$; or 3) invalid nodes with no valid nodes within D hops $\tilde{a}(v) = 0$: $distH_{\text{enr}}(v, q) = distV(v, q) + h$.

Let $Z_1, \dots, Z_{m_\alpha}$ be the distances to nodes with $\tilde{a}(v) = \alpha$. Their distribution is $H(t) = F(t - h(1 - \alpha))$.

Since $h(1-\alpha) < h$, we have: $F(t) \geq H(t) \geq G(t)$. So the distances Z_i to nodes with $\tilde{a}(v) = \alpha$ are stochastically smaller than the distances Y_j to invalid nodes but larger than X_i.

The probability s'' of selecting a valid node is: $s'' = P(X_{(1)} \leq \min\{Y_{(1)}, Z_{(1)}\})$ Since $Y_{(1)}$ is stochastically larger than $Z_{(1)} = \min\{Z_1, \dots, Z_{m_1}\}$: $s'' = P(X_{(1)} \leq Z_{(1)})$. Moreover, given that $F(t) \geq H(t)$, $Z_{(1)}$ is stochastically larger than $X_{(1)}$. Therefore,

$$s'' = P(X_{(1)} \leq Z_{(1)}) \geq P(X_{(1)} \leq Y_{(1)}) = s'.\square$$

Proposition. The ratio R_3 of valid nodes in the observable path under a hybrid distance with enriched attributes is greater than or equal to the ratio R_2 under the hybrid distance alone.

Using the hybrid distance and enrichment process, which gather information about valid nodes up to D hops (with D as the random walk length), our goal is not only to maximize the number of valid nodes in the search path P but increase the expected number of valid nodes in the neighbors N_v^1 for any $v \in P$.

The enrichment via the non-backtracking random walks from a node v can be seen as sampling from the tree T_v^D, rooted at v with depth D. The enriched attribute $\tilde{a}(v)$ correlates with the count of valid nodes within T_v^D.

Proof. By prioritizing navigation towards neighbors with high enriched attribute values $\tilde{a}(v)$, we effectively navigate to nodes v where the number of valid neighbors N_v^1 is maximized. Since v is selected into the search path P, the expected number of valid neighbors N_v is larger than the baseline; thus, $E[N_v^1 \mid v \in P] \geq E[N_v^1]$.

The local valid probability s_{loc} is defined as the probability that a child u of v (i.e., an immediate neighbor in T_v^D) is valid, given that v is on the search path P:

$$s_{\text{loc}} = P(a(u) = 1|v \in P) = \frac{E[N_v^1 \mid v \in P]}{\text{Number of neighbors of } v}$$

Without any conditioning, the expected number of valid neighbors of v is:

$E[N_v^1] = (\text{Number of neighbors of } v) \times s$. Given the selection bias toward nodes with higher N_v, we have: $E[N_v^1|v \in P] \geq (\text{Number of neighbors of } v) \times s$. Therefore, the local valid probability satisfies: $s_{\text{loc}} = \frac{E[N_v^1|v\in P]}{\text{Number of neighbors of } v} \geq s$. The total expected valid nodes in P_{obs} is: $E_{\text{tot}}^{(3)} = p \cdot s'' + p \cdot M \cdot s_{\text{loc}}$ Since $s'' \geq s'$ and $s_{\text{loc}} \geq s$:

$$R_3 = \frac{E_{\text{tot}}^{(3)}}{|P_{\text{obs}}|} = \frac{s'' + Ms_{\text{loc}}}{1+M} \geq \frac{s' + Ms}{1+M} = R_2 \square$$

Time complexity. We now provide a complexity analysis for the RWalks indexing and query answering algorithms. For the sake of clarity, we consider that the base graph is HNSW. The HNSW index builds a multi-layer graph where each node is connected to a bounded number of neighbors within each layer, leading to a construction complexity of $O(n \log n)$, where n is the number of nodes, per the original HNSW paper [47]. RWalks introduces a post-processing enrichment step involving multiple random walks for each node in the base layer. During these random walks, attribute data from nearby nodes are aggregated and transformed into a new average vector using the mean. The number of operations performed during these random walks does not depend on the number of points in the index (the length and number of random walks are fixed), resulting in a constant number of operations per node. Thus, the additional complexity due to this post-processing step is $O(n)$. The combined complexity of building the original HNSW index and applying our post-processing step is $O(n \log n + n)$. Therefore, the dominant term $O(n \log n)$ dictates the overall complexity. For query answering, we preserve the core beam search for which the original HNSW paper [47] reports an average complexity of $O(\log n)$. As our modifications (hybrid distance, distance pruning) described in Alg. 2 are independent of the number of indexed points n, the average search complexity remains $O(\log n)$.

Space complexity. Using graph-based indexes for filtered search incurs an additional space overhead compared to a base graph used for unfiltered search. The cost of building the base graph

depends on the type of graph index used. For example, in the HNSW graph, memory consumption is determined by the storage of graph connections. The number of connections per element is M_0 for the base layer and M for all other layers. Therefore, the average space consumption per element is approximately $(M_0 + m_L \cdot M) \cdot \textit{edgeBitBudget}$ [47] where m_L is the number of layers and *edgeBitBudget* is the number of bits used to represent each edge. The second cost is related to the storage of attribute information. The attribute space ($\mathbb{P}$) is the set of all unique attribute values (p attributes). For instance, if the attributes are *color* and *out-of-stock*, the attribute space includes (green, red, black, and out-of-stock). In real scenarios, the attribute distribution is usually heavy-tailed, with a few frequent attributes and many rare ones, often following a power-law distribution [12]. Thus, the attribute space can have a high cardinality. For example, the Yfcc dataset [7] used in the Filter Track of the NeurIPS BigANN Competition [7] has an attribute space with cardinality 200, 386 even if each data point has only 5 to 15 attributes on average.

Ad-hoc methods have the advantage of being independent of the attribute space cardinality. They do not require storing the node-attribute matrix because, during search, they interface with third-party systems (such as an attribute index from a relational database) to verify that a data point satisfies a filter. Conversely, some hybrid methods like HQANN must store this matrix ($p * n$) as it is used for indexing. This is the case also for RWalks, where the matrix is transformed via diffusion enrichment.

Storing this matrix incurs an important space overhead. Due to the sparse nature of the attribute distribution, we propose to reduce this overhead as follows: instead of storing the full row for each node, requiring $n * p$ values, we store only the active attributes in an unordered map data structure [14], where the key is the attribute ID, and the value is a floating value encoding the attribute enrichment. This reduces storage overhead while supporting efficient lookup, as unordered maps have an average lookup of $O(1)$ and a worst case of $O(p')$ [14], where ($p' << p$) is the average number of active attributes for each node. In addition to the node attributes, RWalks stores attributes aggregated via random walk diffusion. Due to the bounded nature of the walks, the worst-case storage requirement is $D \times m$ additional attributes (constant relative to n), where D is the walk depth and m is the number of walks. This assumes each step introduces unique attributes.

In summary, the average space complexity of RWalks is primarily determined by three factors: storing the graph, the vector data matrix, and the node attributes. Each of these components requires space linearly proportional to the number of nodes n in the graph. Specifically, we store a limited number of edges per node, a matrix with n rows, and an average of p' attributes per node. As a result, the overall average space complexity is O(n).

5 Experiments

5.1 Experimental Framework

Setup. All methods were compiled with GCC 8.2.0 under Ubuntu Linux 20.04 with optimization 3. Experiments were conducted on an Intel(R) Xeon(R) Gold 6242R server (2 sockets, 20 cores per socket, and 2 threads per core, L1d cache: 1.3 MiB, L1i cache: 1.3 MiB, L2 cache: 40 MiB, L3 cache: 71.5 MiB) with a 500GB RAM.

Datasets. We use four real-world datasets: (i) *Deep* [63], containing 1 billion 96-dimensional vectors extracted from CNN layers; (ii) *Sift* [38, 67], with 1 billion 128-dimensional image descriptors; (iii) *Yfcc* [7], comprising 100 million 200-dimensional CLIP embeddings; and (iv) *Arxiv* [55], featuring 1.7 million 384-dimensional paper abstract embeddings. We extract subsets of varying sizes (1M to 100M) from Deep and Sift, labeled by dataset and size (e.g., Sift10M). For Yfcc, we use a 10M subset following the BigANN NeurIPS Competition [7]. We use the real attributes provided with Yfcc (15 attributes/vector) and Arxiv (1 attribute/vector), and we synthetically enrich each vector in Yfcc, Sift and Deep with six categorical attributes (specificities: 1%, 5%, 10%, 20%, 30%, 50%) from 140 categories, as is common in the literature [32, 53]. Note that for Yfcc, we use two datasets, one with the real attributes, called Yfcc10M-real, and another with synthetic attributes, called Yfcc10M, to cover a broad specificity range.

Queries. The Yfcc10M-real query workload includes 100K queries, with 30K having specificity between 1% and 17%. For queries with very low specificity, i.e. < 1%, our experiments (Fig. 8a) show that RWalks and other baselines struggle to achieve recall rates above 90% at acceptable QPS. Following the NeurIPS competition submissions [16, 49], we adopt a linear scan for these low-specificity queries. In Fig. 3a, we report results for the index-only workload (30K queries), and in Fig. 3b, we report results using a linear scan on queries below the cutoff specificity of each method and the index-supported search otherwise (100K queries). For Arxiv, we use the full workload of 5K queries with specificities ranging between 1% and 11%. For Sift, Deep and Yfcc, we construct six workloads with controlled specificity ranging from 1% to 50%, each consisting of 1.6K 10-NN queries (1.6K = 10K queries/6 workloads). Equality filters assign one attribute to each query. AND filters use two attributes. OR filters combine two randomly chosen attributes per query. In the case of NOT filters, each data point is assigned three attributes with varied specificities, and for each query, one attribute is negated.

Metrics. We evaluate the accuracy using *recall@10* and the efficiency using *throughput*, i.e. Queries Per Second (QPS). We also report the indexing *footprint* in GBs and *time* in seconds.

Baselines. We evaluate RWalks against the state-of-the-art approaches. We represent Filter-Then-Search (FTS) with a sequential scan, Search-Then-Filter (STF) with HNSW [47], inline techniques with HNSW-Inline and IVF-Inline, and hybrid techniques with HQANN [77], ACORN-1 [53], ACORN-G [53] and filtered-DiskANN [32]. We use the same HNSW code base from the HNSWLIB library [79] for STF, HNSW-Inline, HQANN and RWalks. For HNSW-Inline, we use the code base as is because it already supports conditions on the addition of points to the result set during graph traversal. For STF, we add a post-filtering step to HNSW's query answering algorithm. For RWalks, we add a post-processing attribute enrichment step on top of the indexing algorithm of two state-of-the-art graph-based indexes NSG [30] (RWalks-NSG) and HNSW (RWalks-HNSW), and implement our own search function following Alg. 2. Unless noted otherwise, we refer to RWalks-HNSW simply as RWalks. Since the HQANN code is not available publicly, we modify the HNSW code by implementing a hybrid distance following the definition in [77]. For Inline-IVF, we use the code from the Faiss library [40], exploiting the IDSelector interface for filtering [57], which provides a list of permitted point IDs to scan based on the query filters. For Filtered-DiskANN, we use the original code [48]. For ACORN, we use the original code [66] built on top of the Faiss implementation of HNSW.

Procedure. Experiments involve two steps: index building and query answering. Caches are fully cleared before each step, and are kept warm within the same batch query workload. Each index is built using 48 threads. All query workloads are executed concurrently in batch mode using 48 threads. Each method is allowed at most 48 hours to build one index.

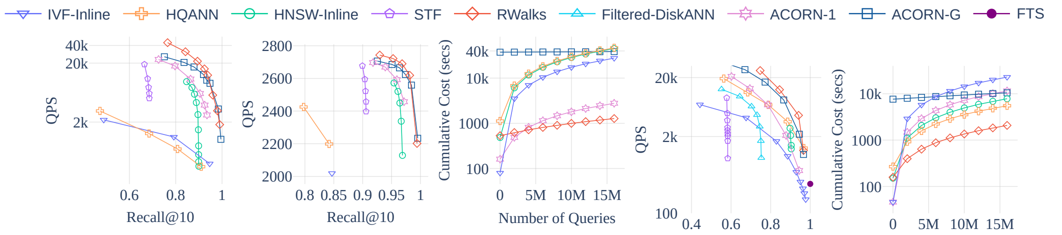
Parametrization. For STF, HNSW-Inline, HQANN, ACORN-G, ACORN-1 and RWalks, we build the HNSW indexing using $M = 32$ and $efConstruction = 200$. As recommended in [53], we set the construction parameter γ to be the inverse of the smallest specificity in the query workload. Then, we vary the search parameters to study the recall-QPS tradeoffs. For RWalks, we use $m = 5$ (number of walks), $D = 3$ (depth of walks). We vary efSearch from 10 to 900 for all methods except for HNSW-Inline where we vary it from 10 to 200 as it saturates at a constant recall using small search parameters. For RWalks search, we use $\tau = 0$, and $h = 0.1$ throughout all experiments, except on Yfcc10M-real, where we use $\tau = 0$ for or $\tau = -1$ (disabling pruning) to obtain the best Recall/QPS curve. For STF, we set $K' = min(10/specificty, ef_{search})$ for two reasons: 1) to ensure accurate results, the search step must return at least $\frac{K}{\text{specificity}}$ candidates allowing us to retain K neighbors after filtering; 2) to provide a more robust baseline than the alternative when we use a fixed k' ($k' = k$) as was done in the Filtered-DiskANN paper [32], where recall saturates naturally at the filter specificity. Note that Filtered-DiskANN does not consider the filter's specificity when choosing K'. For IVF-Inline, the number of clusters is a function of the root square of the number of indexed points as recommended in the Faiss documentation [26]. Thus, we set the number of clusters to $\sqrt{n}$, $2 * \sqrt{n}$, $4 * \sqrt{n}$ and $8 * \sqrt{n}$ for datasets of sizes 1M, 10M, 50M, and 100M respectively where n is dataset size as they lead to the best performance. The number of probes for searching was varied between 1 to 280. For Filtered-DiskANN, we use the same indexing parameters as the original paper ($R = 96$, $LFiltered = 100$, $L = 1$), and we vary the search parameter $LSearch$ between 16 and 2000.

5.2 Query Answering Performance

Performance on Real Datasets. We start with datasets that come with real query workloads, i.e., Yfcc10M-real and Arxiv1.7M. For Yfcc10M-real (Fig. 3), RWalks achieves a QPS@90 of 21K, outperforming ACORN-G (13K) and ACORN-1 (6.1K). Methods like IVF and HQANN perform poorly, with QPS below 0.5K before reaching a 90% recall. Notably, STF stagnates at a 68% recall, and Filtered-DiskANN was excluded for its inability to handle multi-attribute queries. Fig. 3c shows the cumulative cost (indexing + search). The first tick represents the indexing time for each method reaching a 90% recall. While ACORN-G is the second-best competitor, it requires 76x more indexing time than RWalks (37,488s vs. 493s). Though ACORN-1 has the fastest indexing, RWalks surpasses it after answering approximately 3 million queries, as ACORN suffers from low QPS due to the neighbor expansion done during search (ACORN-G performs this on all points during indexing). On the Arxiv dataset, RWalks maintains its superiority with ACORN-G ranking second, and ACORN-1 ranking third with QPS@90 at 1,394s vs. 8,347s for RWalks (Fig. 3d). RWalks wins over competitors, in particular ACORN-1, with a larger margin on the cumulative cost (Fig. 3e). RWalks effectively prioritizes neighbors that guide to valid regions, reducing distance evaluations through multi-hop information, accumulated during indexing as detailed in § 3.3-3.4. In contrast, ACORN-1 only has access to 2-hop neighbors and must evaluate a static number of neighbors in search time to maintain graph traversal, widening the performance gap.

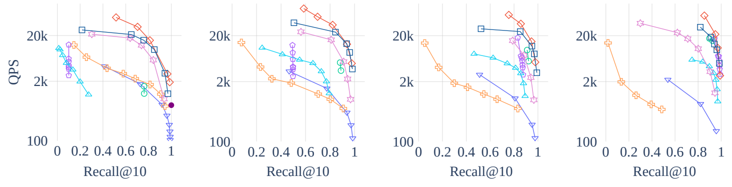
We now evaluate the accuracy/efficiency trade-offs on real datasets using controlled specificity ranging from 1% to 30%. For space considerations, we only report results for Sift10M and Yfcc10M (experiments on Sift1M, Deep1M and Deep10M exhibit the same trends and can be found in [58]). Fig. 4 indicates that RWalks exhibits a good performance consistently across all specificity levels, being faster than the next best method by up to 1.4x on Yfcc10M in achieving a recall of 90%, especially in low-specificity regimes (Fig. 4c). Sift10M shows the same trends as Yfcc10M with even wider gaps (Fig. 6). Note that FTS is an exact search, thus always has a recall of 1 and a fixed value for the QPS. ACORN-G is the next best competitor to RWalks across all specificity levels yet it spends significantly more time building the index (e.g., 2,311s vs. 451s as seen in Fig. 9b). As discussed earlier, ACORN-G and ACORN-1 fall behind RWalks due to the neighborhood expansion step performed during indexing and search respectively. HNSW-Inline and HQANN do not have a stable performance across different specificity values. HNSW-Inline experiences high latency in low-specificity regimes (Figs. 4a and 6a) due to extended traversal times, leading to more distance computations, as we explained in § 3.1. Conversely, HQANN's QPS at 90% recall decreases as specificity increases because it needs to probe larger graph clusters of points with similar attributes (Fig. 6). Another key observation is that HNSW-Inline exhibits recall saturation, particularly in low-specificity regimes. For instance, at 1% specificity, recall saturates at 80% and 76% on Sift10M (Fig. 6a) and Yfcc10M (Fig. 4a) respectively. STF, which uses HNSW, suffers from low recall when specificity is low (Figs. 4a and 6a). This is because vector search does not yield enough valid neighbors. Recall that the number of retrieved points in the search step is defined as $k' = min(ef_s, k/s)$, where s is the query specificity. Thus, even if k' is higher in low-specificity settings, retrieved points may validate the query filters but not be close to the query. However, in high-specificity regimes, where filters are more relaxed, recall levels are met, and STF performs similarly to RWalks (Figs. 4d and 6c) in terms of QPS, due to the absence of additional data structures or computations in the search process (e.g., hybrid distance). Filtered-DiskANN uses a pruning strategy to prioritize edges between points with similar attributes. It shows inverse trends compared to HQANN where performance improves as specificity increases.

Robustness Under Different Filter Types and Specificities at High Recall. We evaluate QPS at a 90% recall across specificities from 0.1% to 50%. We use four filter types: a single equality filter



(a) Yfcc10M-Real (s>1%) Query (b) Yfcc10M-Real Query (c) Yfcc10M-Real Index (d) Arxiv Query (e) Arxiv Index

Fig. 3. Query/Index performance with real filters (Yfcc10M-Real, Arxiv)



(a) 1% (b) 5% (c) 10% (d) 30%

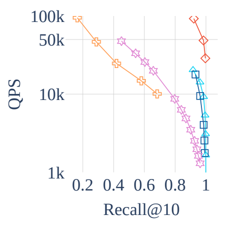
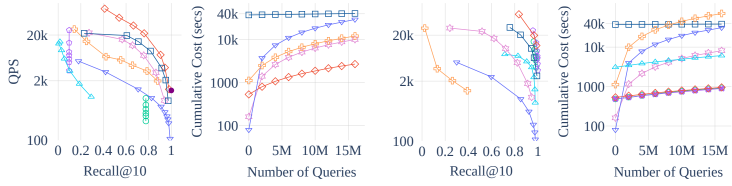


Fig. 4. Query performance with varied specificity (Yfcc10M) **Fig. 5. Unfiltered search (Sift1M)**



(a) 1% Query (b) 1% Index (c) 30% Query (d) 30% Index

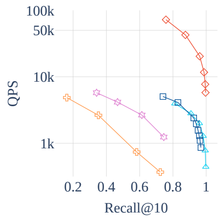
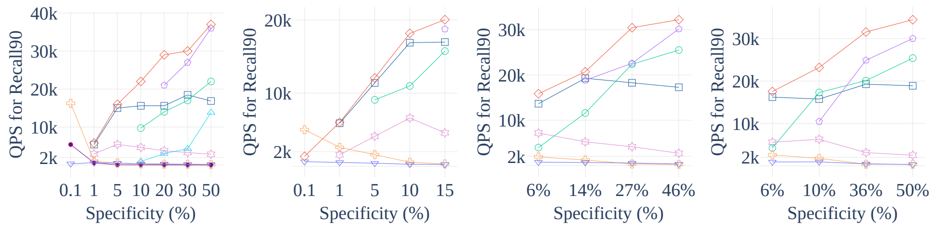


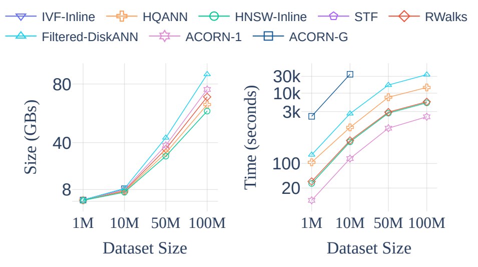
Fig. 6. Query/Index performance with varied specificity (Sift10M) **Fig. 7. Unfiltered search (Sift50M)**



(a) Equality filters (b) AND filters (c) OR filters (d) NOT filters

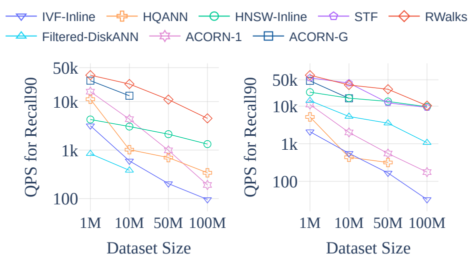
Fig. 8. Query performance with different filters (Yfcc10M)

(Fig. 8a), an AND filter combining two attributes (Fig. 8b), an OR filter on two attributes (Fig. 8c) and a NOT filter (Fig. 8d). We omit methods unable to reach a 90% recall, and include Filtered-DiskANN in the equality filter evaluation only as it lacks support for composite filters. RWalks outperforms other methods by up to 1.41x in low-specificity regimes (e.g., 10%) and maintains high QPS across a wide range of specificities (10%–30%). HQANN, excels at 0.1% specificity, but quickly falls behind once specificity exceeds 1%. At extremely low specificities (0.1%), methods like STF, HNSW-Inline, RWalks and ACORN struggle to achieve a 90% recall. Note, though, that a linear scan, rather than an index, would be the method of choice in this case. We observe that methods are impacted differently by this limitation. For instance, RWalks achieves a 68% recall at 770 QPS (a linear scan achieves a 5,400 QPS with a 100% recall), whereas HNSW-Inline and STF saturate below a 5% recall. As for ACORN, even with $\gamma = 1000$ (the inverse of the 0.1% specificity), recall saturates at 8%. Filtered-DiskANN achieves a 90% recall only at specificities higher than 10%. Its QPS improves as specificity increases, yet it still lags behind methods built upon the HNSW graph. We observe the same trends on composite AND filters (Fig. 8b), with a few exceptions. HQANN's QPS increases compared to single filters with similar specificity. This is because HQANN uses a hybrid distance when building the graph, so nodes sharing the same attributes tend to be connected. In the case of AND filters, more attributes are valid, which results in shorter graph traversals and a higher QPS. RWalks is the only method, other than HQANN, that can achieve a 90% recall at very low specificities (0.1%). In extremely low specificities (<= 0.1%), ad-hoc methods suffer from low recall. This is expected since their performance correlates positively with specificity. As for RWalks, it achieves a 90% recall at 0.1% specificity in AND filters (Fig. 8b) but not in equality filters. This is due to two factors: 1) during the enrichment process, the bounded depth of the walks may prevent capturing attributes at such low specificity, and 2) while the overall specificity of AND filters is low, it is derived from the product of the specificities of its individual attribute pairs which are higher, and thus captured more effectively during the enrichment process (e.g., the AND filter of specificity 0.1% is the product of the specificities 10% and 1%). Figs. 8c and 8d show results for the OR and NOT filters (the latter is modeled as an OR filter in the case of RWalks). Trends are similar to equality filters with a small QPS decrease due to processing more than one attribute in the hybrid distance compared to the equality filter. In conclusion, RWalks remains the most robust method across varied specificity levels.



(a) Footprint **(b) Time**

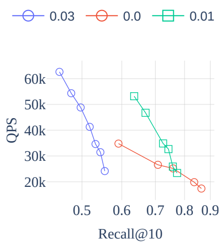
Fig. 9. Index construction overhead with dataset size (Sift)



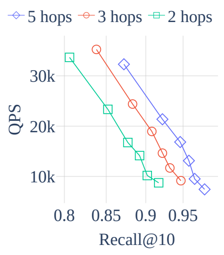
(a) 5% specificity **(b) 30% specificity**

Fig. 10. Query performance with dataset size (Sift, Recall@10 = 0.9)

Scalability with Dataset Size at High Recall. We now evaluate query scalability with increasing dataset size using the Sift dataset (1M to 100M). Fig. 11 shows the highest QPS at a 90% recall for two specificities. Methods unable to reach a 90% recall or complete indexing within 48 hours are excluded. When specificity is low, RWalks outperforms all methods due to its distance pruning heuristic and the graph's logarithmic complexity. At high specificity, STF, although unable to meet



(a) τ



(b) D

Fig. 11. Query performance with different random walks depth (D) and τ

Table 2. Indexing overhead (Time/Footprint)

Method	Time (seconds)			Footprint (GB)		
	Yfcc10M	Yfcc-Real	Arxiv-1.7M	Yfcc10M	Yfcc-Real	Arxiv-1.7M
HNSW	483	483	151	9.3	9.3	3.1
Disk-ANN	3094	-	268	14.9	-	4.53
ACORN-1	161	161	46	10.8	10.8	1
ACORN-G	37488	37488	7636	15.8	15.8	3.5
IVF	80	80	46	8.1	8.1	2.7
HQANN	1316	1140	268	9.5	9.7	3.1
RWalks	493	533	158	10.0	10.3	3.2

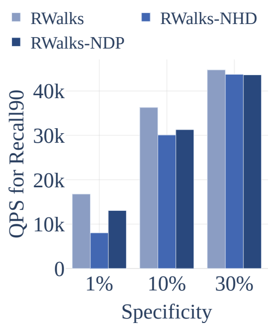


Fig. 12. Ablation study

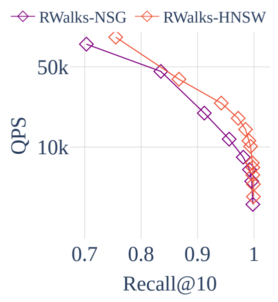


Fig. 13. RWalks-NSG vs. RWalks-HNSW (Sift1M)

the recall criteria when specificity is low, performs efficiently (Fig. 10b), benefiting from relaxed filters and no additional search computations. HQANN exhibits a sharper QPS decline, dropping 16x between Sift1M and Sift50M, compared to a 4.41x average drop for RWalks, HNSW-Inline, and STF. On larger datasets like Sift100M, HQANN's recall saturates below 90%. A similar pattern is seen in the ACORN family as shown in Fig. 11, where the QPS gap between ACORN and RWalks widens as the dataset grows. This is due to ACORN's neighbor expansion and its alternative to RNG-based pruning, which, while improving filtering, negatively affects search performance.

Versatility Under Different Query Types. A key advantage of ad-hoc methods, which leave the index intact and modify only the search process, is that they retain the original purpose of the index, i.e., unfiltered vector search. In contrast, hybrid methods that alter index construction (e.g., HQANN integrates attributes in connecting graph nodes and ACORN changes the pruning heuristic) limit its utility for unfiltered search. RWalks, however, supports both filtered and unfiltered search efficiently by adding a post-processing step after index construction that preserves the base index (i.e., node connections remain unchanged). This allows RWalks to perform filtered search more efficiently than competitors, while matching HNSW's performance in unfiltered search. Fig. 5 compares hybrid methods (RWalks, Filtered-DiskANN, ACORN-G, ACORN-1, HQANN) on unfiltered queries. Due to HQANN's low QPS in high-specificity regimes, it shows naturally low QPS for unfiltered queries (equivalent to 100% specificity). HNSW-Inline and STF were excluded as they are equivalent to RWalks. We further evaluate performance on a larger dataset (Sift50M), where building separate indexes for filtered and unfiltered search can be impractical due to memory constraints. In Fig. 7, we use indexes built for filtered search (specificity $\geq$ 8%) to answer unfiltered search. We observe that RWalks is up to 13x faster than the next best competitor (ACORN-G). These results confirm that RWalks is the only method that efficiently supports both filtered and unfiltered search using the same base index.

Impact of RWalks Parameters. RWalks depends on three parameters: the number of random walks m, their depth D, and a search pruning factor τ. Attribute enrichment is achieved by performing m random walks of length D per node. Longer walk depths (D) improve recall, particularly in low-specificity regimes, with a marginal increase in indexing time. Fig. 11b illustrates the impact of D on the recall-QPS tradeoff at a specificity of 5%, where higher values of D result in increased recall at the same QPS, as longer walks help locate nodes with low-specificity attributes. In high-specificity regimes (>20%), we observed no significant impact, as attributes with high specificity are typically captured in earlier hops. Due to space constraints, results for high-specificity regimes are provided in the supplementary archive [58]. Next, we analyze the effect of D on indexing time, which comprises graph construction and attribute enrichment. Since D determines the length of random walks, it primarily affects the attribute enrichment phase. On the Sift1M dataset (Fig. 11b), attribute enrichment takes 3.72s, 3.74s, and 3.79s for $D = 2$, $D = 3$, and $D = 5$ respectively, compared to an average total indexing time of 36.3s. This shows that attribute enrichment accounts for approximately 10.1% of the total indexing time (7.9% when working with larger datasets such as Sift10M). For m, increasing the number of random walks m enhances the likelihood of finding a given attribute due to the randomness of the walks. Naturally, higher values of m increase the enrichment time. On the Sift1M dataset, enrichment takes 1.1s for $m = 1$ and 3.79s for $m = 5$ (respectively 2.9% and 10.1% of the total indexing time). Beyond $m = 5$, recall-QPS curves stabilize, showing only marginal improvements in recall. Recall-QPS results for different m values are included in our archive [58]. Throughout our experiments, we use $D = 3$ and $m = 5$ because they led to good trade-offs overall. In § 3.4, we introduced the distance pruning heuristic controlled by τ, which adjusts the ratio of evaluated neighbors during graph traversal. Increasing τ improves QPS but may lead to recall saturation, as fewer candidates are available to continue the search. Fig. 11a shows the trade-offs between QPS and recall for different values of τ (0, 0.01, 0.03). With $\tau = 0.03$, QPS reaches 53K with 60% recall, compared to 32K for $\tau = 0$. However, recall saturates at 55% with higher τ. We recommend using $\tau = 0$ to maximize QPS while maintaining recall above 90%. For cases requiring very high recall (e.g., 95%-99%) or hard datasets, such as with Yfcc10M-real, disabling pruning entirely avoids recall saturation and boosts recall to 99%. On all other datasets, we achieved high recall levels with pruning at $\tau = 0$.

5.3 Indexing Performance

As discussed earlier, ad-hoc methods like FTS and STF struggle across different specificity regimes (Figs. 4a, 4b and 6a). To improve search performance, hybrid methods introduce extra overhead during indexing. ACORN, for instance, builds a denser graph where node degree correlates with the lowest query specificity, while HQANN uses a hybrid distance metric during indexing to optimize for low-specificity queries (Fig. 8a). However, HQANN struggles with high-specificity queries, and Filtered-DiskANN, which leverages attribute-based neighborhood diversification, only performs well in high-specificity regimes. RWalks overcomes these limitations by preserving the base vector search index and adding a lightweight post-processing step for attribute enrichment (Alg. 1). We compared the scalability of RWalks with ACORN, HQANN, and Filtered-DiskANN in terms of indexing time and footprint, using the Sift dataset (1M to 100M). Fig. 9b shows that RWalks incurs only a 9% time overhead over HNSW due to attribute enrichment. In contrast, ACORN-G achieves similar search performance but requires significantly longer indexing times (e.g., 34, 410s on Sift10M vs. 451s for RWalks), and failed to complete indexing for larger datasets within 48 hours.

ACORN-1, an approximation of ACORN-G, is faster at indexing (even compared to HNSW) but slow at search due to its avoidance of RNG-based pruning. HQANN adds substantial overhead (e.g., 4, 753s more than RWalks for Sift50M) because its graph construction requires hybrid distance calculations, slowing down the insertion operations. RWalks, on the other hand, benefits from an easily parallelizable post-construction process, with independent random walk generation, attribute aggregation, and write operations, without any locking. This results in a consistent time overhead across all dataset sizes (e.g., 3.7s to enrich 1M points). Filtered-DiskANN also incurs higher indexing costs (e.g., 2, 613s for Sift10M, 2, 163s more than RWalks). In terms of memory footprint, both HQANN and RWalks store attributes in memory to support search and indexing. HQANN uses attributes for the hybrid distance, while RWalks employs them for both the hybrid distance and the pruning heuristic. As shown in Fig. 9a, RWalks has a slightly higher memory footprint than HQANN (+9%), reaching up to 6GB for Sift100M. ACORN incurs additional memory overhead due to its high-degree graph structure (up to 1.5x). Moreover, since attributes are not stored directly in the index, external systems (e.g., a relational database index) are needed to annotate valid points based on query filters during search. In our experiments, we simulated this with a bit vector, though this was not included in ACORN's query time evaluation.

5.4 RWalks with Different Graph Indexes

A key feature of RWalks is that it is agnostic to the base index on which it is applied. In previous experiments, we implemented RWalks on top of HNSW because it is a popular state-of-the-art index for unfiltered vector search. In this experiment, we implement RWalks on top of NSG [30, 72], another strong baseline. Fig. 13 demonstrates that RWalks is equally effective on both data structures, the variations being due to the nature of the base graphs themselves. The consistent performance of RWalks across these graphs can be attributed to two factors: 1) the distribution of attributes remains independent of the graph structure; and 2) state-of-the-art vector search methods, including HNSW and NSG, typically exploit the same beam search algorithm during query answering.

5.5 Ablation Study

As described in § 3, the RWalks search process consists of three main building blocks (the twin result set, the hybrid distance and the pruning heuristic).

We disable each of these components individually and evaluate the performance of RWalks, using three specificity levels (1%, 10% and 30%). Fig. 12 shows the QPS required to achieve a 90% recall with the full RWalks (Alg. 2), compared to RWalks-NDP, i.e., RWalks without the distance pruning heuristic (line 10 in Alg. 2) and RWalks-NHD, i.e., RWalks without the hybrid distance (line 13 in Alg. 2).

6 Conclusions

In this paper, we propose RWalks, a graph-based ng-approximate vector search method that efficiently supports filtered and unfiltered vector search. RWalks achieves this thanks to three key ideas: 1) a hybrid distance; 2) an index-agnostic post-indexing attribute enrichment process; and 3) a distance pruning heuristic. We demonstrate the superiority, versatility and robustness of RWalks with an extensive experimental comparison. In the future, we plan to generalize RWalks to support range queries and other distance measures.

Acknowledgments

This work used the UM6P African Supercomputing Center (ASCC) and was supported by an Oracle PhD Fellowship.

References

[1] 2019. Hnswlib - fast approximate nearest neighbor search. https://github.com/nmslib/hnswlib.

[2] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. HD-index: Pushing the Scalability-accuracy Boundary for Approximate kNN Search in High-dimensional Spaces. *PVLDB* 11, 8 (2018).

[3] Martin Aumuller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* (2020).

[4] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. Elpis: Graph-Based Similarity Search for Scalable Data Science. *PVLDB* 16, 6 (2023).

[5] 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, Article 43 (2025), 31 pages. doi:10.1145/3709693

[6] A. Babenko and V. Lempitsky. 2015. The Inverted Multi-Index. *TPAMI* 37, 6 (2015).

[7] Big ANN Benchmarks. 2024. YFCC100M 10M Slice for Filtered Vector Search Benchmarking. https://github.com/harsha-simhadri/big-ann-benchmarks/tree/main/neurips23/filter. Accessed: 2024-07-23.

[8] Andreas Blattmann, Robin Rombach, Kaan Oktay, Jonas Müller, and Björn Ommer. 2022. Retrieval-augmented diffusion models. *Advances in Neural Information Processing Systems* 35 (2022), 15309–15324.

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

[10] Paolo Ciaccia and Marco Patella. 2000. PAC Nearest Neighbor Queries: Approximate and Controlled Search in High-Dimensional and Metric Spaces. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, David B. Lomet and Gerhard Weikum (Eds.). IEEE Computer Society, 244–255. doi:10.1109/ICDE.2000.839417

[11] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*.

[12] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703.

[13] Douglas Comer. 1979. Ubiquitous B-tree. *ACM Computing Surveys (CSUR)* 11, 2 (1979), 121–137.

[14] cppreference.com. 2024. C++ Reference: std::unordered_map Container. https://en.cppreference.com/w/cpp/container/unordered_map. Accessed: 2024-07-23.

[15] Wei Dong. 2022. KGraph: Open-Source Library for k-NN Graph Construction and Nearest Neighbor Search. http://www.kgraph.org. Accessed: 2024-07-23.

[16] Matthijs Douze. 2024. YFCC - FAISS baseline. https://github.com/harsha-simhadri/big-ann-benchmarks/tree/main/neurips23/filter/faiss.

[17] Karima Echihabi. 2020. High-Dimensional Similarity Search: From Time Series to Deep Network Embeddings. In *SIGMOD*.

[18] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2022. Hercules Against Data Series Similarity Search. *PVLDB* 15, 10 (2022).

[19] Karima Echihabi, Themis Palpanas, and Kostas Zoumpatianos. 2021. New Trends in High-D Vector Similarity Search: AI-driven, Progressive, and Distributed. *Proc. VLDB Endow.* 14, 12 (2021), 3198–3201. http://www.vldb.org/pvldb/vol14/p3198-echihabi.pdf

[20] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2020. Big Sequence Management: on Scalability. In *IEEE BigData*.

[21] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. Big Sequence Management: Scaling up and Out. In *EDBT*.

[22] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. High-Dimensional Similarity Search for Scalable Data Science *(ICDE)*.

[23] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2018. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *PVLDB* 12, 2 (2018).

[24] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2019. Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. *PVLDB* 13, 3 (2019).

[25] Andreas Engel, Rémi Monasson, and Alexander K Hartmann. 2004. On large deviation properties of erdös–rényi random graphs. *Journal of Statistical Physics* 117, 3 (2004), 387–426.

[26] Faiss. 2024. Faiss Wiki: Guidelines for Choosing an Index. https://github.com/facebookresearch/faiss/wiki/Guidelines-to-choose-an-index. Accessed: 2024-07-23.

[27] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2000. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the ninth international conference on Information and knowledge management.* 202–209.

[28] Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228* (2016).

[29] Cong Fu, Changxu Wang, and Deng Cai. 2021. High Dimensional Similarity Search with Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[30] 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 (Jan. 2019), 461–474. doi:10.14778/3303753.3303754

[31] K Ruben Gabriel and Robert R Sokal. 1969. A new statistical approach to geographic variation analysis. *Systematic zoology* 18, 3 (1969), 259–278.

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

[33] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware Locality-sensitive Hashing for Approximate Nearest Neighbor Search. *PVLDB* 9, 1 (2015), 1–12.

[34] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondrej Chum. 2019. Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5070–5079.

[35] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A survey on locality sensitive hashing algorithms and their applications. *arXiv preprint arXiv:2102.08942* (2021).

[36] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *Advances in Neural Information Processing Systems*, Vol. 32.

[37] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *TPAMI* 33, 1 (2011).

[38] Herve Jegou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: Re-rank with source coding. In *ICASSP*.

[39] Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, and Xiaofei He. 2014. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics* 44, 11 (2014), 2167–2177.

[40] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* (2017).

[41] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906* (2020).

[42] Jon Kleinberg et al. 2002. Small-world phenomena and the dynamics of information. *Advances in neural information processing systems* 1 (2002), 431–438.

[43] Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. 2020. Pre-training via paraphrasing. *Advances in Neural Information Processing Systems* 33 (2020), 18470–18481.

[44] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[45] 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.

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

[47] Yury 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* (2018), 824–836.

[48] Microsoft. 2024. DiskANN code base. https://github.com/microsoft/DiskANN. Accessed: 2024-07-23.

[49] MingYang. 2024. YFCC10M Filtered Search Benchmark: DISKANN Baseline Configuration. https://github.com/harsha-simhadri/big-ann-benchmarks/blob/main/neurips23/filter/fdufilterdiskann/config.yaml. Accessed: 2024-07-23.

[50] Marius Muja and David G. Lowe. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP International Conference on Computer Vision Theory and Applications*. 331–340.

[51] Javier Vargas Munoz, Marcos A Gonçalves, Zanoni Dias, and Ricardo da S Torres. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition* 96 (2019), 106970.

[52] Themis Palpanas. 2020. Evolution of a Data Series Index: the iSAX Family of Data Series Indexes. *CCIS* 1197 (2020).

[53] 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. doi:10.1145/3654923

[54] Alexander Ponomarenko, Yury Malkov, Andrey Logvinov, and Vladimir Krylov. 2011. Approximate nearest neighbor search small world approach. In *International Conference on Information and Communication Technologies & Applications*, Vol. 17.

[55] Qdrant. 2024. Qdrant: Benchmark Datasets for Filtered Approximate Nearest Neighbor Search. https://github.com/qdrant/ann-filtering-benchmark-datasets. Accessed: 2024-07-23.

[56] D. Raj. Reddy. 1977. Speech Understanding Systems: A Summary of Results of the Five-Year Research Effort. *Department of Computer Science Technical Report. Carnegie Mellon University* (1977).

[57] Facebook AI Research. [n. d.]. Searching in a subset of elements. https://github.com/facebookresearch/faiss/wiki/Setting-search-parameters-for-one-query#searching-in-a-subset-of-elements. Accessed: 2024-06-26.

[58] RWalks. 2024. RWalks additional material and codebase. https://github.com/anon-sigmod/RWalks/tree/main. Accessed: 2024-07-23.

[59] Hanan Samet. 1990. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[60] Roger W Schvaneveldt, Francis T Durso, and Donald W Dearholt. 1989. Network structures in proximity data. In *Psychology of learning and motivation.* Vol. 24. Elsevier, 249–284.

[61] Lei Shi. 2013. Trading-off among accuracy, similarity, diversity, and long-tail: a graph-based recommendation approach. In *Proceedings of the 7th ACM Conference on Recommender Systems.* 57–64.

[62] Harsha Vardhan Simhadri, George Williams, Martin Aumuller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2022. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. *CoRR* abs/2205.03763 (2022). doi:10.48550/arXiv.2205.03763 arXiv:2205.03763

[63] Skoltech Computer Vision. 2018. Deep billion-scale indexing. http://sites.skoltech.ru/compvision/noimi.

[64] Liuyihan Song, Pan Pan, Kang Zhao, Hao Yang, Yiming Chen, Yingya Zhang, Yinghui Xu, and Rong Jin. 2020. Large-scale training system for 100-million classification at alibaba. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 2909–2930.

[65] Suhas Jayaram Subramanya, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems.* 13766–13776.

[66] TAG-Research. [n. d.]. ACORN codebase. https://github.com/TAG-Research/ACORN. Accessed: 2024-10-02.

[67] TEXMEX Research Team. 2018. Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr/.

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

[69] Godfried T Toussaint. 2002. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface* 34 (2002).

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

[71] Mengzhao Wang, Li ang Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2022. Navigable Proximity Graph-Driven Native Hybrid Queries with Structured and Unstructured Constraints. arXiv:2203.13601

[72] 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 (jul 2021), 1964–1978. doi:10.14778/3476249.3476255

[73] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. 2013. A data-adaptive and dynamic segmentation index for whole matching on time series. *Proceedings of the VLDB Endowment* 6, 10 (2013), 793–804.

[74] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.

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

[76] Elmar Wolfstetter, Uwe Dulleck, Roman Inderst, Peter Kuhbier, and M Lands-Berger. 1993. *Stochastic dominance: theory and applications.* Humboldt-Univ., Wirtschaftswiss. Fak.

[77] 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 amp; Knowledge Management (CIKM '22).* ACM. doi:10.1145/3511808.3557610

[78] Yan Xia, Kaiming He, Fang Wen, and Jian Sun. 2013. Joint Inverted Indexing. *ICCV* (2013).

[79] Yury Malkov. [n. d.]. Header-only C++/python library for fast approximate nearest neighbors. https://github.com/nmslib/hnswlib. Accessed: 2024-07-23.

[80] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining.* 635–644.

[81] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1979–1991.

Received October 2024; revised January 2025; accepted February 2025