Adaptive Query-Aware Hybrid Search in Vector Databases

Adeel Aslam
University of Southampton
United Kingdom
A.Aslam@soton.ac.uk

Rizwan Khan
Dundalk Institute of Technology
Ireland
rizwan.khan@dkit.ie

Giovanni Simonini
University of Modena and Reggio Emilia
Italy
simonini@unimore.it

George Konstantinidis
University of Southampton
United Kingdom
G.Konstantinidis@soton.ac.uk

Abstract

Hybrid search, which integrates vector and structured retrieval, is essential for efficient and accurate information access over large-scale data in modern AI-based applications. We build upon HNSW, a state-of-the-art approximate nearest neighbor index for efficient hybrid search that organizes data in multi-layer proximity graphs. We exploit information from previously executed queries to inform new ones to start with the right foot—i.e., by selecting more effective entry points for the proximity graph exploration. This strategy accelerates convergence from the earliest search steps and improves accuracy. Finally, we experimentally evaluate our approach on six diverse datasets under varying settings, demonstrating consistent improvements.

Keywords

Vector databases, HNSW, Hybrid query, Binary search tree, Recall

1 Introduction

Combining lexical and semantic retrieval has become essential for many applications, including e-commerce, clinical information retrieval, and legal or compliance search [16, 25, 26]. To address this need, several vector databases have integrated hybrid search capabilities that unify keyword-based and embedding-based retrieval, such as Weaviate [39], Milvus [24], and PostgreSQL [29].

Vector databases store unstructured data as vector embeddings, enabling efficient similarity search [25]. These vectors are typically produced by models such as BM25 [32], BERT [8], or large language model–based embedding generators (e.g., OpenAI's text-embedding models[1]) which capture the semantic meaning of text [10]. Each data item is often associated with structured attributes (e.g., category, age, or limit), allowing hybrid queries that combine semantic similarity with structured filtering [7].

To efficiently identify nearest neighbors in large vector collections, Approximate Nearest Neighbor (ANN) search techniques are widely used. In ANN search, the dataset is first indexed to organize data points, after which queries are executed against this index [2]. Common indexing methods include quantization [42], inverted file indexing [4], and proximity graphs [3, 38], with the latter offering an effective balance of accuracy and efficiency [26].

Proximity graphs, including HNSW, NSW, NSG, and Vamana, are widely used in the literature for vector search [11, 14, 23, 25]. However, directly applying proximity graphs such as HNSW in hybrid retrieval remains challenging. Hybrid search, like any search, must simultaneously optimize two often competing objectives, accuracy and efficiency; hybrid search must do this while operating over distinct data modalities: structured attributes and unstructured embeddings, all within a single query [7].

[1] https://platform.openai.com/docs/guides/embeddings

Hybrid indexing. Existing studies employ two main methodologies for leveraging proximity graphs in hybrid search: vector-based graphs (Graph) and attribute-based graphs (Attr), as summarized in Table 1. Vector-based approaches refer to constructing indices solely based on a distance function, without any attribute considerations. These are the most efficient approaches for vector similarity queries but need to be adapted for hybrid queries, generally requiring higher-order graph traversal (e.g., two or multi-hop searches) to identify items that meet the query predicates [28, 34]. In attribute-based hybrid search, *dedicated graphs* are constructed for particular attributes or for specific predicate operations, e.g., by using pre-filtering for point queries [6], or by sorting the dataset according to attribute values for range queries [41, 43]. While attribute-based methodologies achieve efficient run-time retrieval of nearest neighbors with high accuracy, creating these indices can be computationally expensive, and moreover requires a very large memory overhead, which can explode in the face of an increasing number of attributes.

In this paper, we are interested in an efficient and accurate hybrid search that is also memory efficient. Instead of building a large number of different graphs like in attribute-based approaches, we would like to support a range of hybrid queries by loading just one graph, agnostic to attributes or predicate operators of the query. For this, we are going to use a naïve vector-based graph, such as the HNSW graph [23], and build an adaptive and efficient graph traversal mechanism based on historical (past) queries. This will allow us to achieve high accuracy while simultaneously considering the query predicates.

Our contribution. In existing vector-based graph methods and their hybrid-search variants, the entry point for a new query is typically chosen heuristically–for example, by starting from a fixed node or selecting a random node and traversing toward its nearest neighbors. Yet, the selection of entry points in proximity graphs–particularly in the HNSW graph at `level-0` during the initial steps of search–plays a crucial role in determining search efficiency. Starting with the right foot allows the search to quickly move toward the most relevant region of the graph, thereby accelerating convergence and improving accuracy. For instance, we can select the top-K entry points corresponding to previously executed similar queries to guide an efficient search over the graph structure for a new query. Motivated by this, we propose a new hybrid search method that reuses information from previously executed similar queries to guide entry point selection, by leveraging the idea of progressively building the index for vector search based on the query [21].

Overall, our contributions can be summarized as follows: (1) We propose a novel adaptive, query-aware hybrid search approach that considers both vectors and attributes for efficient exploration of graph-based data structures (Section 3). (2) We introduce a method to efficiently organize queries based on distance, attributes, and recall, enabling fast and high-quality identification of entry points for high-order graph traversal (Section 3.1). (3) We provide a mechanism to handle point, range, conjunction,

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.42

and disjunction query predicates distinctly, ensuring accurate and efficient search performance while maintaining agnosticism (Section 3.2). (4) We conduct extensive experiments on real-world datasets, comparing the proposed approach against recent agnostic and dedicated state-of-the-art methods, as well as existing vector database (Section 4).

2 Preliminaries

In this section, we provide a discussion on several preliminaries for the proposed adaptive query-aware hybrid search.

2.1 Hybrid search

Hybrid search retrieves the nearest neighbor vectors for a query while also satisfying specific predicate conditions—such as point or equality (`name = 'ABC'`), range (`age BETWEEN 10 AND 40`), conjunction (name = 'ABC' AND age BETWEEN 20 AND 50), or disjunction (category = 'Sports' OR views > 1000)—on query attributes (e.g., age, category, views). Pre-filtering is generally more effective for smaller datasets or high-selectivity queries, whereas graph-based approximate nearest neighbor search is commonly used for low-selectivity queries, where a large portion of the dataset satisfies the predicate condition [23, 28].

2.1.1 Hierarchical navigable small world. HNSW is an advanced and widely employed indexing data structure from the family of proximity graphs for ANN search [23]. It is a multi-layer graph structure inspired by the traditional skip list [27]. The bottom layer is the densest, containing all nodes interconnected, while the upper layers contain progressively fewer nodes as the hierarchy ascends. The connectivity of the graph is controlled by the parameter M, which introduces a trade-off between efficiency and accuracy. The search process begins from the top layer at a random entry point and greedily navigates towards lower layers, where the actual nearest neighbors are explored based on the distance function. Many modern vector databases employ HNSW for efficient ANN search [24, 30, 39].

2.1.2 Nearest neighbor expansion. In HNSW graph exploration, the bottom layer plays a crucial role as it contains all nodes, with connections established based on proximity [34]. Graph-based approaches leverage higher-level exploration, such as two-hop neighbors from the most appropriate node at `level-0`, to find nearest neighbors that also satisfy predicate conditions [28, 34].

2.2 Binary search tree

In this study, we employ a standard Binary Search Tree (BST)-based approach to efficiently organize the similar queries that share the same attribute [17]. This organization is subsequently used for entry point identification in the dense HNSW graph. The insertion process follows the standard BST paradigm [17], in which each node is placed according to a computed score that incorporates both distance information (specifically, the nearest distance among the top-K distances for a query) and recall as detailed in Section 3.1.3.

During the search phase, the tree is traversed starting from the root. At each step, a query-dependent score is dynamically computed to capture the similarity between the query embedding and the embedding stored at the tree nodes, together with the estimated recall. Insertion relies on the proximity between a query and the nearest node distance among the top-K items, whereas search is based on query and past queries in the BST nodes; both similarity measures are derived from the same embedding space and are therefore correlated with execution similarity. Moreover, we do not rely solely on distance; we also take recall information into account when evaluating query similarity. For most queries, all retrieved entry points are also valid entry points in the graph that satisfy the query attribute predicate.

Table 1: Comparison of vector-based graph and attribute-based hybrid indexing techniques. Notation: M, M_γ = connections; N = vectors; R = higher-order traversal; L, F = labels and attributes; V = visited nodes; $|P[L]|$, $|t(q)|$ = per-label vectors and query tags; Pre = preprocessing. Q_p= Set of queries for predicate p

Technique	Graph	Attr.	Search Complexity	Construction Complexity
HNSW PF [7]	✓	✗	$O(M \log N)$	$O(NM \log N)$
ACORN-1 [28]	✓	✗	$O((M+R)\log N)$	$O(NM \log N)$
ACORN-γ [28]	✓	✗	$O((M_\gamma + R)\log N)$	$O(NM_\gamma \log N + R \log N)$
Navix [34]	✓	✗	$O((M+R)\log N + \lvert k_s \rvert)$	$O(NM \log N)$
UNG [6]	✗	✓	$O(L) + O(M \log N)$	$O(M\lvert F \rvert^2 L) + O(Pre)$
TAG [20]	✗	✓	$O(V \cdot M \cdot \lvert t(q) \rvert)$	$O(\lvert t(p) \rvert + \sum_{t \in t(p)} \lvert P[L] \rvert \cdot M)$
IRange [41]	✗	✓	$O(M \log N)$	$O(NM \log_b N) + O(N \log N)$
SERF [43]	✗	✓	$O(M \log N)$	$O(NM \log N) + O(N \log N)$
Query-aware (ours)	✓	✓	$O((M + R)\log N) + O(\log_2 Q_p)$	$O(NM \log N) + O(Pre)$

2.3 Recall estimation

Typically, similarity assessment relies solely on proximity measures. However, to identify the most appropriate query from the set of previously executed queries, we also incorporate recall information alongside distance metrics to ensure high quality.

For unseen queries, we estimate recall using features such as the nearest query distance, computed from a small sample of points in the dense HNSW graph structure, and query selectivity. Recall is then predicted using a *linear regression model*, which computes it as a weighted sum of these features plus a bias term [13].

3 Query-aware search

We propose a query-aware approach that employs a graph-based structure, such as HNSW, to handle queries with different predicates.

3.1 Adaptive query-aware hybrid search

The entire procedure for the proposed query-aware top-K hybrid search is outlined in Algorithm 1. This algorithm takes as input a query (q) and the layer-0 of the HNSW graph ($\mathcal{I}_{hnsw}$(Layer-0)) with graph exploration factor ef_{search}. The properties of the query include the query embedding (V), attribute (A) or bin number in case of range query, predicate condition (P), the required number of results (K), selectivity (σ), and the estimated average distance (d_{near}) between q and data points.

3.1.1 Searching of BST. The algorithm first identifies the BST associated with the attribute (A) and predicts the estimated recall (R_{est}) for the query q using a regression model with the nearest distance (d_{near}) and query selectivity (σ) as input features (lines 1–3). To compute d_{near}, a small set of random data points is sampled from the dense HNSW structure, and SIMD instructions are used to accelerate the distance computations between the query (q) and these points, with the smallest distance selected as d_{near}. After obtaining this prediction, the algorithm traverses the corresponding tree using the R_{est} and the V (line 4). Tree traversal is guided by a score that combines recall and the normalized distance between the V and the tree node embedding

$(n.v)$ ($\texttt{score}_{search}$: $w_1 * (1 - \frac{\text{dist}(n.v,V)}{D_{\max}}) + w_2 * R_{est}$). Here, $D_{\max}$ denotes the maximum distance, which is computed offline by sampling data points and determining the largest observed distance for normalization. The relative importance of distance and recall in the scoring function is controlled by the weights (w) as detailed in Section 4.

3.1.2 HNSW graph traversing. During traversal of the tree from the root toward the most appropriate node, multiple entry points are accumulated (line 4). These entry points correspond to the index positions in the HNSW graph of the top-K nearest neighbors from previously executed queries stored in the BST. These index positions serve as entry points for high-order graph traversal, such as two-hop search [28, 34] with graph exploration property ef_{search}, enabling the identification of the nearest neighbors for the q that also satisfy the P (line 5).

3.1.3 Node insertion in BST. The result of the query q is used to update the associated BST for the corresponding attribute and to train and improve the prediction accuracy of the regression model (lines 6–8). For insertion, the score of a new tree node is computed based on the normalized nearest-neighbor distance (d_{nn}) of the top-K items retrieved after HNSW graph exploration, along with the actual recall, using the same weights applied during the search phase (line 7). The insertion process follows the standard BST insertion procedure based on the computed score. Each node in the tree stores the properties of the query (q), as well as the index positions of the top items (based on ef_{search}) retrieved after graph exploration for that q, and $\texttt{score}_{insert}$ for tree exploration. Additionally, the average distance d_k among the top-K retrieved nearest neighbors (sampled during search to estimate the nearest distance for q), and the query selectivity (σ) are used as input features, while the query's actual recall is used as the output label to train the linear regression model.

In cold-start settings and for queries with extreme selectivity (very high or very low), a pre-filtering or post-filtering step is applied based on query selectivity (σ). For low-selectivity queries, we employ post-filtering, as it enables faster processing while maintaining good accuracy. In contrast, for high-selectivity queries, we rely on pre-filtering, since it ensures full recall with only a small overhead in execution time. This information is also used to update the associated BST for the attribute and the regression model.

3.2 Predicates evaluations

The proposed query-aware approach can handle several predicate operators, including point, range, conjunction, and disjunction operators.

3.2.1 Point predicate. For point predicate queries, we build a BST based on the query attribute. This attribute is used to determine the query selectivity from the dataset. For low-selectivity queries, where a large portion of the data satisfies the predicate, a dedicated BST is initialized for the corresponding attribute. In contrast, for high-selectivity queries, a separate tree is not constructed. All entry points retrieved from the BST already satisfy the point predicate condition and are used as entry points for higher-order graph traversal.

3.2.2 Range predicate. A recent study states that supporting every possible query range requires $O(n^2)$ dedicated graphs, which is conceptually similar to maintaining separate indexes for each range [41]. However, constructing and maintaining a separate BST for every possible range is prohibitively expensive. To address this limitation, we employ an equi-bin partitioning strategy over different ranges of a particular data field, which can be used to support range predicate queries.

For range predicates, we adopt an equi-bin methodology based on the Freedman–Diaconis rule [33] to determine an optimal bin size. The data field used for range predicates is partitioned using a bin width $h = 2 \times \frac{\text{IQR}(x)}{n^{1/3}}$, where h denotes the bin width, $\text{IQR}(x)$ is the interquartile range of the data, and n is the total number of data points. The number of bins is then computed as $m = \frac{\max(x) - \min(x)}{h}$ and every bin has upper and lower bound that defines the range of values it contains.

Each bin corresponds to a dedicated tree structure that is initialized in advance. For queries spanning multiple ranges, consecutive bins maintain references to adjacent trees, enabling efficient processing of long-range queries. Moreover, the search across these connected trees can be easily parallelized. After traversal, the nearest nodes (index positions on the graph) obtained from all relevant queries are used as entry points for higher-order graph exploration. In this study, we adopt a post-filtering strategy when selectivity exceeds 40%.

3.2.3 Conjunction and disjunction queries. We adopt a combined methodology for queries involving conjunction (`AND`) and disjunction (`OR`) operators between point and range predicates. Such queries explore the relevant BSTs corresponding to the query ranges as well as the BSTs associated with equality predicate attributes. The resulting explorations yield multiple entry points, which are subsequently used for high-order graph traversal.

3.3 Analysis

We assume that all executed queries share the same memory footprint as in other state-of-the-art methodologies. However, in our approach, we maintain these queries within a lightweight data structure that can be easily loaded into memory, as this process occurs before HNSW graph traversal. For a given query, the regression model uses the nearest-neighbor distance from a global sample at inference time, while during training it relies on the average distance of the top-K retrieved neighbors, which captures local neighborhood structure and correlates with the inference distance.

Moreover, we employ a lightweight learning model, where the training (learning) cost is handled offline, and the inference cost is $O(1)$. Similarly, the cost of insertion and traversal within the

Algorithm 1 Adaptive Query-Aware Hybrid Search

Require: Query $q = \langle V, A, P, \sigma, K, d_{near} \rangle$, maximum distance D_{max}, HNSW index $\mathcal{I}_{hnsw}$ (Layer-0) and ef_{search}
Ensure: Top-K nearest neighbors satisfying predicate P
1: **if** Map$[A] \neq \varnothing$ **then**
2: $BST \leftarrow$ Map$[A]$ ▷ BST associated with attribute A or predicate range
3: $R_{est} \leftarrow$ regression_model.predict(d_{near}, σ) ▷ Estimated recall
4: $entryPoints \leftarrow$ traverseBST(BST, V, R_{est})
5: $neighbors$(top-K) $\leftarrow$ twoHopSearch$(\mathcal{I}_{hnsw}, V, A, P, entryPoints)$
6: Compute actual recall R_{act} for q and nearest distance d_{nn}
7: $\texttt{score}_{insert} \leftarrow w_1\left(1 - \frac{d_{nn}}{D_{max}}\right) + w_2 R_{act}$ ▷ w_1 and w_2 are weights for distance and recall, respectively (see Section 4)
8: BST.insert$(\langle q,$ indexPositions$(neighbors)\rangle, \texttt{Score}_{insert})$
9: regression_model.update$(\langle d_k, \sigma \rangle, R_{act})$
10: **else**
11: $neighbors \leftarrow$ pre/post-filterHNSW(q) ▷ Selectivity-based filtering
12: Map$[A] \leftarrow$ new BST$(q,$ indexPositions$(neighbors), \texttt{Score}_{insert})$
13: regression_model.update$(\langle d_k, \sigma \rangle, R_{act})$
14: **end if**
15: **return** Top-K nearest neighbors

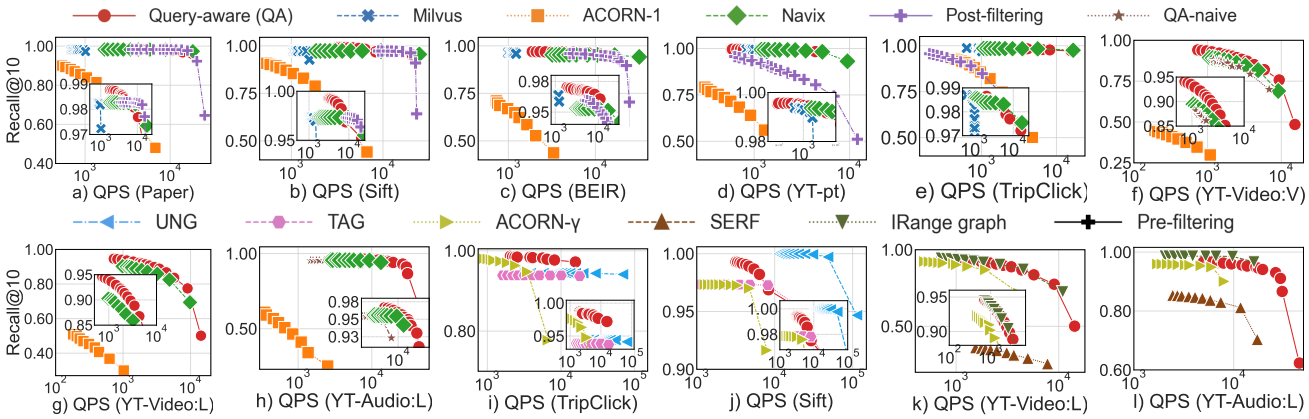

Figure 1: Comparison of point (a-e) and range (f-h) queries for agnostic and dedicated point (i,j) and range (k,l) techniques.

binary tree is $O(\log Q)$ as the score for both insertion and traversing is completely random (not in order), where Q represents the total number of queries, with a constant-time to identify the relevant tree structure. Furthermore, the memory consumption for maintaining such a structure in memory is $O(|A_f|.T)$, where T denotes the number of trees and $|A_f|$ represents the total number of frequent attributes or bins.

4 Experimental Setup and Results

In this section, we provide a discussion on experimental setup, comparators, and results.

Datasets. In this work, we employ six different datasets, as listed in Table 2 (YT-category is the same as YT-video; only the attribute differs), along with their respective properties. In this study, query selectivity is measured relative to the maximum selectivity observed for each attribute in the dataset, which is estimated from attribute frequencies or precomputed equi-bin counts. A query is classified as high-selectivity if the difference from the maximum exceeds 0.8, because it preserves many attributes, enabling high-order graph traversal and achieving high recall.

Comparison approaches. To evaluate the proposed query-aware solution [2], we compare it against several state-of-the-art hybrid search approaches. These include both *agnostic* graph-based methods (Navix [12], Post-filtering, (and pre-filtering, particularly for very high selectivity queries), Milvus [24], ACORN-1 [18], and ACORN-γ, which extend graph structures, and *dedicated* hybrid approaches (UNG [5], TAG [35], SERF [19], and IRange [40]).

For agnostic methods, we use identical graph construction parameters ($M = 64$, $ef_{\text{construction}} = 400$). For dedicated approaches, we adopt the experimental settings reported in their respective studies, except for IRange, which we configure similarly to our agnostic setup, since it also employs `hnswlib` [22].

We vary the ef_{search} parameter from 20 to 1500 with increments of 60, 80, or 120, resulting in 15 points. Each point in the *queries per second* (QPS)-recall trade-off graph shows performance for the corresponding ef_{search} value. Larger ef_{search} values result in higher recall but lower QPS, whereas smaller values enable faster graph exploration at the cost of reduced recall.

For point queries, we assign equal weights to distance and recall (0.5 each) during BST node traversal. In contrast, for range, conjunction, and disjunction queries, the recall weight is divided by the number of BSTs involved in the exploration (i.e., 0.5/total BSTs). This adjusted weighting is used for score computation during both BST traversal and new query insertion into the BSTs.

Experimental setup: We implement our query-aware solution using `hnswlib` for its extensibility and optimization. While some works use FAISS [9], both libraries provide similarly optimized HNSW implementations with SIMD acceleration. All experiments are conducted on an Intel Xeon E5-2697 (2.40 GHz, 72 cores) machine with 216 GB of RAM running Ubuntu 18.04. Each search experiment utilizes 40 threads for parallel execution.

[2] https://github.com/AdeelAslamSoton/QueryAwareIndexing

Table 2: Dataset properties used in experiments.

Dataset	Dim.	Avg. Q. Sel	Size & Queries	Attributes
TripClick [31]	768	0.29	1.14M & 1K	Clinical area
YT-category [1]	1024	0.11	1.29M & 1.5K	Genre
YT-Video [1]	1024	0.33& 0.35	1.29M & 1.5K	views & like
YT-Audio [1]	128	0.33 & 0.35	1.29M & 1.5K	views & like
BEIR [36]	1024	0.28	121M& 1.16K	dataset_name
Paper [37]	200	0.083	2.0M & 10K	Random int (1 to 12)
SIFT [15]	128	0.083	1.0M & 10K	Random int (1 to 12)

4.1 Results and discussion

In this section, we present the results for the adaptive query-aware hybrid search strategy and provide a thorough comparison (some QPS/Recall trade-off graphs also have a zoomed-in view to highlight high recall and QPS values).

4.1.1 Point queries. Figures 1a-1e illustrate the results of point query evaluation and comparison with simple graph-based approaches. Figure 1a presents the results for the Paper dataset. The proposed approach achieves up to 8× higher efficiency than Milvus for high recall (greater than 95%), 30× than ACORN-1, and 6× than Navix. However, at very high recall levels, the query-aware method is only 1.6× more efficient than post-filtering, primarily due to the low selectivity of most Paper queries. A similar performance trend is observed for the SIFT dataset, as illustrated in Figure 1b.

Figures 1c-1e present the results for the BEIR, YouTube, and TripClick datasets. For the BEIR dataset, the query-aware solution outperforms Milvus by 10×, post-filtering by 2×, and Navix by 1.8×, with an even greater improvement over ACORN-1 for high recall, as shown in Figure 1c. For the YouTube dataset, it achieves a 10× improvement over post-filtering, a 1.2× gain over Navix,

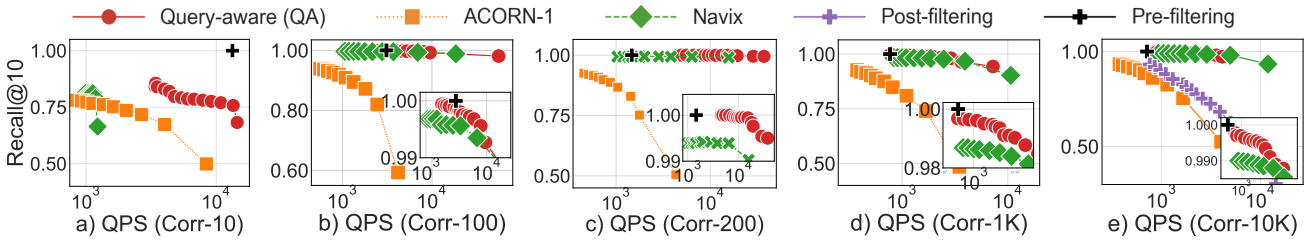

Figure 2: Varying query–data correlation (a–e).

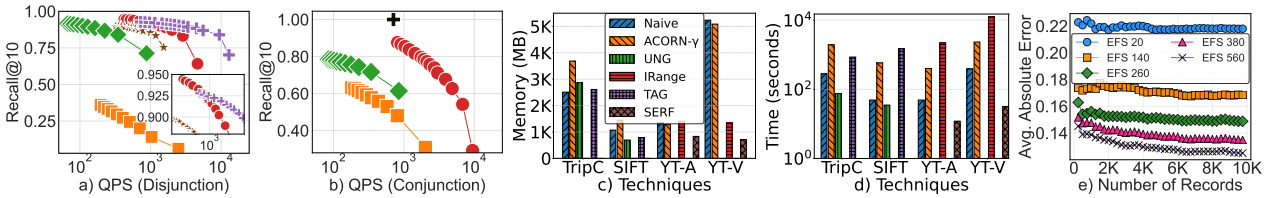

Figure 3: Disjunction and conjunction queries (a,b), memory and time consumption (c,d), and average absolute error (e).

and a 1.4× gain over Milvus, as depicted in Figure 1d. Similarly, for the TripClick dataset, which exhibits slightly high selectivity queries, the query-aware solution is 40× faster than post-filtering and 8× faster than Milvus, whereas ACORN-1 is slow and fails to reach 95% recall, as shown in Figure 1e.

Overall, both the proposed query-aware approach and Navix perform well for high-selectivity queries, as they leverage high-order graph exploration from level-0 entry points. However, for low-selectivity queries, the proposed approach is more efficient and achieves better recall by avoiding extra computations, while Navix incurs overhead from computing local selectivity.

4.1.2 Range queries. Figures 1f– 1h illustrate the results of agnostic approaches for range searches on the YouTube dataset. Across all cases, the proposed query-aware approach significantly outperforms ACORN-1. Specifically, Figure 1f, which presents results for the YT-video dataset using `view` as the range attribute, shows that the query-aware method achieves a 2× speedup over Navix for recall above 90%. Similarly, for Figure 1g, which considers the `like` attribute, the query-aware approach achieves a 2.4× improvement over Navix, and in Figure 1h with audio embeddings, it demonstrates a 1.5× performance gain over Navix for recall above 95%.

For range searches, the proposed approach leverages query selectivity, opting for post-filtering instead of high-order graph traversal for queries with very low selectivity. This two-pronged methodology enhances overall performance, as long-range queries increase the likelihood of more data satisfying the query predicates. Additionally, we construct BSTs for range attributes using an equi-bin strategy, which efficiently determines the entry points in the graph for high-order traversal.

4.1.3 Query-aware and dedicated solutions for point search . Figure 1i and Figure 1j compare the proposed technique with dedicated and extended graph-based solutions for point queries using TripClick (768 dim) and SIFT (128 dim) datasets. For the larger size embedding dataset, UNG only reaches 94% recall and TAG 93% with increasing search parameter, while query-aware achieves 98% recall at 2053 QPS with M=64 and ef_{search}=1500 as shown by Figure 1i. Although ACORN-γ attains higher recall, query-aware is nearly 1.6× faster at high recall and 4× faster at low recall. Similarly, for SIFT, UNG performs better in terms of QPS and recall as depicted by the Figure 1j; however, employing UNG in real-world use cases is challenging because it is only suited for point-predicate filtering.

4.1.4 Query-aware and dedicated solutions for range search. Figures 1k and 1l compare the query-aware approach with dedicated and extended graph-based solutions for range predicate queries on the YouTube dataset, using the `like` attribute for both video (1024-dim) and audio (128-dim) embeddings. Figure 1k shows that the proposed query-aware approach achieves QPS and recall comparable to the iRange graph for video embeddings, while both methods outperform ACORN-γ and SERF. For smaller embedding sizes, the iRange graph attains higher recall and better search performance than the query-aware approach. However, as shown in Figure 1l, the proposed method still demonstrates 2× and 4× performance improvements over ACORN-γ and SERF, respectively. In contrast, iRange graph and SERF require dataset sorting based on the property attribute and the construction of a separate index for each range attribute.

4.1.5 Varying query selectivity and correlation. Figure 2a-2e compare the proposed query-aware approach with graph-based agnostic techniques on the TripClick dataset, across varying query-dataset correlations and selectivities. For these experiments, the dataset is sorted by decreasing query-item distance, and attributes are assigned to data points based on distance. For example, in *correlation-10*, the first and last 10 points are assigned the query attribute, resulting in a selectivity of 20/total items.

Figure 2a for correlation-10 indicates that the query-aware approach achieves 4× higher efficiency than Navix at 80% recall (Navix uses a fixed entry point to `layer-0`, similar to HNSW-PF), while pre-filtering (PF) performs better due to fewer distance computations. Increasing correlation to 100 improves overall recall for higher ef_{search}, however, PF remains superior for high recall as shown in Figure 2b. Moreover, the query-aware approach still outperforms Navix by 1.2× and ACORN-1 by 20×. For correlation-1000, PF efficiency declines as more predicates are satisfied and requires more distance computation, as depicted by Figure 2c. Finally, for correlation-10,000, where each query predicate matches 20K items, the proposed approach is 2× faster

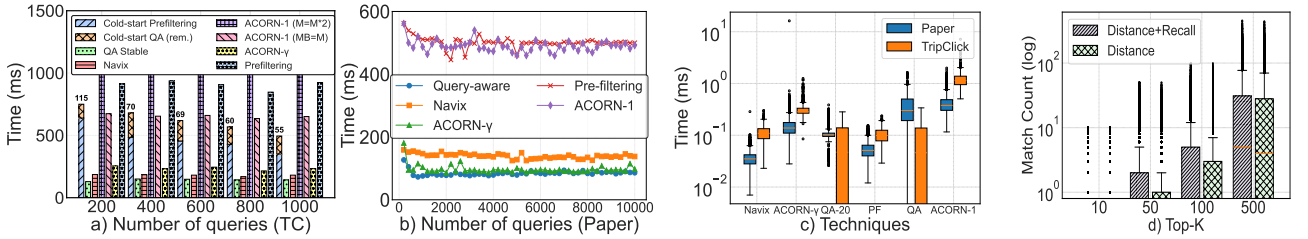

Figure 4: Impact of PF (a, b), impact of tree and graph levels traversal (c), and impact of distance and recall (d).

than Navix and 8× faster than ACORN-1, as shown in Figure 2d. The main advantage of our approach is that it buffers closely related queries, enabling efficient selection of entry points for attribute-based graph exploration.

4.1.6 Disjunction and conjunction queries . Figures 3a and 3b show the performance of disjunction and conjunction queries on the YouTube dataset, using video embeddings with `YT-category` (equality) and `view` (range) attributes. We compare our method against agnostic graph-based baselines. Disjunctive queries exhibit low selectivity, whereas conjunctive queries are highly selective. For disjunctive queries, our approach achieves up to 20× speedup at high recall (>93%), while post-filtering is more effective at lower recall due to low selectivity and reduced graph traversal overhead as depicted by Figure 3a. For conjunctive queries, pre-filtering performs better at high recall due to very high selectivity queries. Overall, our method outperforms ACORN-1 and Navix by up to 12× and 1.8×, respectively, as shown in Figure 3b.

4.1.7 Memory and time consumption. Figures 3c and 3d compare memory consumption and index construction time of the naïve HNSW index (query-aware approach) with $M = 64$ and $efc = 400$ against dedicated (point and range) and extended (ACORN-γ) methods across datasets. For point predicates, on TripClick the naïve approach uses 1.46×, 1.2×, and 2.31× less memory than UNG, TAG, and ACORN-γ, respectively, as shown by Figure 3c. For SIFT dataset, UNG and TAG are slightly more memory-efficient due to smaller embeddings. For range predicates, the naïve index uses less memory than iRange on small embeddings (YT-audio) but slightly more than SERF; for larger embeddings, dedicated methods are more memory-efficient. However, dedicated approaches require separate indexes for each range attribute.

Figure 3d presents the index construction time for dedicated and QA approaches. To ensure a fair comparison, we exclude preprocessing costs for dedicated methods so that the results reflect only graph construction time. For small embedding vectors, the naïve solution requires only 49–52 seconds, significantly outperforming ACORN$_\gamma$, iRange, and TAG. In contrast, for larger embedding vectors, dedicated approaches achieve up to 2× faster construction than the naïve solutions. However, they require separate construction and more pre-processing overhead for every range attribute.

4.2 Ablation study

4.2.1 Regression model accuracy. Figure 3e shows the average absolute error of the linear regression model, which estimates the recall for batches of 200 tuples across different efs. Each point in the graph corresponds to the average absolute error for a single batch. The error peaks at 0.23 for $efs = 20$ and decreases to 0.14 for larger efs (sample size for distance estimation equals efs).

4.2.2 Impact of pre-filtering on proposed solution. Figure 4a (TripClick) (*Cold start prefiltering* number represents tuples processed with prefiltering) and Figure 4b (Paper) show the effect of pre-filtering (cold start) on query processing for 560 efs, measured per batch of 200 tuples. For the TripClick dataset, which contains more distinct attributes, a significant portion of the time is spent on pre-filtering across batches. Once stabilized, the approach outperforms Navix, ACORN-1, and ACORN-γ by 1.2×, 8×, and 2.2×, respectively. Similarly, for the Paper dataset, the query-aware approach initially takes slightly longer due to pre-filtering of new queries, but it remains 4×, 1.43×, 4.2×, and 1.2× faster than ACORN-1, ACORN-γ, pre-filtering alone, and Navix, respectively.

4.2.3 Graph levels vs. tree traversal. Figure 4c compares the graph traversal time of HNSW and BST tree nodes for both the Paper and TripClick datasets. In this experiment, we also tested a quantized version of the embeddings for tree traversal, where 20% of each vector was quantized. For a small number of queries and more distinct attributes (TripClick), tree traversal is faster than competing methods. However, as the number of queries grows (Paper dataset), the BST size and depth increase. In this case, the naive greedy approach for graph traversal from level m to 0 proves to be 1.8× more effective than naive tree traversal.

4.2.4 Impact of distance and recall metrics. Figure 4d illustrates the impact of using both recall and distance versus distance alone on the retrieval quality for varying top-K items. We measure how many executed queries already contain the nearest neighbor of a new query before performing an exhaustive search. The results show that incorporating recall alongside distance retains, on average, 1.6× more items similar to the nearest neighbors compared to using distance alone.

5 Conclusion

In this study, we propose a query-aware solution for agnostic hybrid search. In the proposed approach, each query contributes to improving future queries by identifying better entry points within the dense graph structure, enabling more efficient and accurate retrieval of nearest neighbors. The results of this study reveal that the proposed method outperforms alternative graph-based and agnostic approaches in graph exploration.

Acknowledgments

This research is partially funded by the Horizon Europe projects DATAPACT (No. 101189771) and RAISE Suite (No. 101188337), and the UKRI Horizon Europe Guarantee Scheme for projects UPCAST (No. 101093216) and RAISE (No. 101058479). Additionally, this work is also supported by the CINECA Grant ISCRA B id-HP10BTKGNQ_C.

References

[1] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. 2016. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675* (2016).

[2] Ilias Azizi. 2024. Vector Search on Billion-Scale Data Collections. *Proceedings of the VLDB Endowment. ISSN* 2150 (2024), 8097.

[3] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-based vector search: An experimental evaluation of the state-of-the-art. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–31.

[4] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.

[5] YZ Cai. 2025. Unified Navigating Graph (UNG). https://github.com/YZ-Cai/Unified-Navigating-Graph. Accessed: 2025-10-08.

[6] Yuzheng Cai, Jiayang Shi, Yizhuo Chen, and Weiguo Zheng. 2024. Navigating labels and vectors: A unified approach to filtered approximate nearest neighbor search. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–27.

[7] Yannis Chronis, Helena Caminal, Yannis Papakonstantinou, Fatma Özcan, and Anastasia Ailamaki. 2025. Filtered Vector Search: State-of-the-Art and Research Opportunities. *Proceedings of the VLDB Endowment* 18, 12 (2025), 5488–5492.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[9] Facebook Research. 2025. Faiss. https://github.com/facebookresearch/faiss

[10] Kayla Farivar. 2025. Semantic Search for Information Retrieval. *arXiv preprint arXiv:2508.17694* (2025). https://arxiv.org/abs/2508.17694

[11] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).

[12] Gaurav. 2025. Faiss-Navix. https://github.com/gaurav8297/faiss-navix. Accessed: 2025-10-08.

[13] Andrew Jacobsen and Ashok Cutkosky. 2024. Online Linear Regression in Dynamic Environments via Discounting. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. 21083–21120.

[14] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems* 32 (2019).

[15] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. 2010. Aggregating local descriptors into a compact image representation. In *IEEE computer society conference on computer vision and pattern recognition*. 3304–3311.

[16] Guoxin Kang, Zhongxin Ge, Jingpei Hu, Xueya Zhang, Lei Wang, and Jianfeng Zhan. 2025. BigVectorBench: Heterogeneous Data Embedding and Compound Queries are Essential in Evaluating Vector Databases. *Proceedings of the VLDB Endowment* 18, 5 (2025), 1536–1550.

[17] Michał Komorowski and Tomasz Trzciński. 2019. Random binary search trees for approximate nearest neighbour search in binary spaces. *Applied Soft Computing* 79 (2019), 87–93.

[18] Guestrin Lab. 2025. ACORN. https://github.com/guestrin-lab/ACORN. Accessed: 2025-10-08.

[19] Rutgers DB Lab. 2025. SeRF. https://github.com/rutgers-db/SeRF. Accessed: 2025-10-08.

[20] Jiarui Luo, Miao Qiao, Chaoji Zuo, and Dong Deng. 2025. Tag-Filtered Approximate Nearest Neighbor Search. In *International Conference on Data Engineering*. 3642–3654. doi:10.1109/ICDE65448.2025.00272

[21] Vasilis Mageirakos, Bowen Wu, and Gustavo Alonso. 2025. Cracking Vector Search Indexes. *Proceedings of the VLDB Endowment* 18, 11 (September 2025), 3951–3964. doi:10.14778/3749646.3749666

[22] Yury Malkov. 2025. hnswlib: Hierarchical Navigable Small World graphs. https://github.com/nmslib/hnswlib. Accessed: 2025-10-08.

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

[24] Milvus Contributors. 2025. Milvus Documentation. https://github.com/milvus-io/milvus-docs Original date: 2020-05-27T09:12:23.

[25] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *The VLDB Journal* 33, 5 (2024), 1591–1615.

[26] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Vector database management techniques and systems. In *Companion of the 2024 International Conference on Management of Data*. 597–604.

[27] Thomas Papadakis. 1993. *Skip lists and probabilistic analysis of algorithms*. University of Waterloo Ph. D. Dissertation.

[28] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. Acorn: Performant and predicate-agnostic search over vector embeddings and structured data. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[29] pgvector contributors. 2025. pgvector: Open-source vector similarity search for Postgres. https://github.com/pgvector/pgvector Accessed: 2025-10-06.

[30] Pinecone-VDB. 2025. Pinecone - vector database. https://www.pinecone.io/

[31] Navid Rekabsaz, Oleg Lesota, Markus Schedl, Jon Brassey, and Carsten Eickhoff. 2021. TripClick: The Log Files of a Large Health Web Search Engine. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2507–2513. doi:10.1145/3404835.3463242

[32] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[33] Daniel D Saurette, Asim Biswas, Adam W Gillespie, et al. 2024. Determining minimum sample size for the conditioned Latin hypercube sampling algorithm. *Pedosphere* 34, 3 (2024), 530–539.

[34] Gaurav Sehgal and Semih Salihoglu. 2025. NaviX: A Native Vector Index Design for Graph DBMSs With Robust Predicate-Agnostic Search Performance. In *Proceedings of the 51st International Conference on Very Large Data Bases (VLDB)*. ACM, 4438–4450. doi:10.14778/3749646.3749704

[35] SpaceIshtar. 2025. FilterGraph (TAG). https://github.com/SpaceIshtar/FilterGraph. Accessed: 2025-10-08.

[36] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=wCu6T5xFjeJ

[37] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2022. Navigable proximity graph-driven native hybrid queries with structured and unstructured constraints. *arXiv preprint arXiv:2203.13601* (2022).

[38] Mengzhao Wang, Haotian Wu, Xiangyu Ke, Yunjun Gao, Yifan Zhu, and Wenchao Zhou. 2025. Accelerating Graph Indexing for ANNS on Modern CPUs. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–29.

[39] Weaviate-VDB. 2025. Filtered Vector Search | Weaviate - vector database. https://weaviate.io

[40] Yuexuan Xu. 2025. iRangeGraph. https://github.com/YuexuanXu7/iRangeGraph. Accessed: 2025-10-08.

[41] Yuexuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S. Jensen. 2024. iRangeGraph: Improvising Range-Dedicated Graphs for Range-Filtering Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 6 (December 2024), 26 pages.

[42] Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. 2025. Pqcache: Product quantization-based kvcache for long context llm inference. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–30.

[43] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 1 (March 2024), 26 pages. doi:10.1145/3639324