Elastic Index Selection for Label-Hybrid AKNN Search

Mingyu Yang[1,2], Wenxuan Xia[1], Wentao Li[3], Raymond Chi-Wing Wong[2], Wei Wang[1,2]
[1]The Hong Kong University of Science and Technology (Guangzhou),
[2]The Hong Kong University of Science and Technology, [3]University of Leicester
{myang250,wxia248}@connect.hkust-gz.edu.cn,wl226@leicester.ac.uk,raywong@cse.ust.hk,weiwcs@ust.hk

ABSTRACT

Real-world vector embeddings often carry additional label attributes, such as keywords and tags. In this context, label-hybrid approximate k-nearest neighbor (AKNN) search retrieves the top-k approximate nearest vectors to a query, subject to the constraint that each result must contain labels from a specified label set. A naive solution involves building a separate index for each label set; however, the exponential number of combinations renders this approach storage-intensive and impractical. To overcome this, we propose selectively indexing only a subset of label sets while still ensuring efficient query processing for all possible queries. This is made possible by a key insight into label containment: an index built for label set L can serve any query with a superset L', with query efficiency bounded by the elastic factor—the ratio between the number of vectors matching L and those matching L'. We formalize index selection as a constrained optimization problem that aims to determine which label sets to index in order to satisfy space and query efficiency constraints. We prove the problem is NP-complete and propose efficient greedy algorithms for its efficiency- and space-oriented variants. Extensive experiments on real-world datasets show that our method achieves 10×–800× speedups over state-of-the-art techniques. Moreover, our approach is index-agnostic and can be seamlessly integrated into existing vector database systems.

PVLDB Reference Format:
Mingyu Yang, Wenxuan Xia, Wentao Li, Raymond Chi-Wing Wong, Wei Wang. Elastic Index Selection for Label-Hybrid AKNN Search. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/mingyu-hkustgz/LabelANN.

1 INTRODUCTION

The k**-nearest neighbor (KNN)** search over high-dimensional vectors is a core operation in modern data systems, underpinning a wide range of applications such as recommendation systems [42], data mining [6], face recognition [50], product search [50], and retrieval-augmented generation (RAG) for large language models [32]. In real-world deployments, vector embeddings are often associated with **label attributes**, such as product brands, keywords, and geolocations. For example, in an e-commerce setting, a user may search for items similar to a given photo while specifying label constraints like brand name and release year.

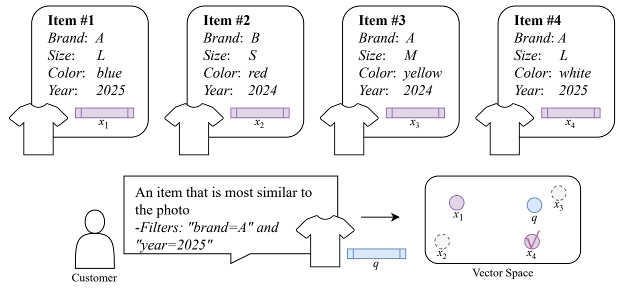


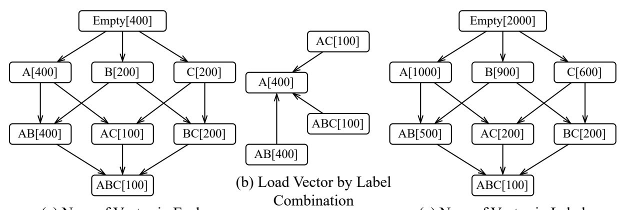
Figure 1: The Example of Label-Hybrid AKNN Search

Example 1. *Fig. 1 shows a label-hybrid AKNN search in an online shopping scenario. Each item, represented by an embedding vector* $x_1, \ldots, x_4$*, is associated with label attributes such as brand and year. Customers search for items using a reference photo q along with label constraints. The goal is to retrieve the item most similar to q that satisfies all label constraints. Accordingly,* x_1 *and* x_3 *are excluded due to label mismatch, and* x_4 *is returned as the nearest neighbor.*

In this context, we introduce the problem of label-hybrid k-nearest neighbor (KNN) search, illustrated in Fig. 1. Unlike traditional KNN search, the label-hybrid variant considers both the vector embeddings and associated label attributes of each data point. Formally, each entry in the database S consists of a vector embedding and a label set L. Given a query vector q and a query label set L_q, the goal is to retrieve the top-k nearest neighbors of q in S, where each returned vector must contain all labels in L_q, i.e., $L_q \subseteq L$. However, due to the curse of dimensionality [26], computing exact label-hybrid KNN in high-dimensional spaces is computationally expensive. Consequently, recent research has focused on the label-hybrid approximate KNN (AKNN) search problem, which aims to efficiently return the top-k approximate nearest neighbors that satisfy the label constraint, achieving significant performance gains by trading off a small amount of accuracy [5, 40].

Existing Solutions. To support label-hybrid AKNN search, existing approaches often employ graph-based indexes [10, 11, 19, 21, 24, 28, 33, 34, 37, 41, 45, 48] combined with filter-based search strategies, due to their good search efficiency. Specifically, the graph-based index is constructed by adding base vectors in database S as nodes and connecting each node to its nearby neighbors to form edges, which are carefully selected to support efficient navigation. At query time, on top of the graph-based index, filter-based search strategies [19] are applied to enforce label constraints and retrieve the final results. In particular, two strategies—PreFiltering and PostFiltering—are integrated into the graph-based index. The PreFiltering strategy prunes nodes (i.e., base vectors) that do not

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



(a) Num of Vector in Each Label Group (b) Load Vector by Label Combination (c) Num of Vector in Label Containing Group

Figure 2: The Motivating Example

Example 2. *In Fig. 2, we illustrate a vector database S where labels are drawn from the alphabet $\{A, B, C\}$. Vectors are grouped by identical label sets L, with each group denoted as L[number of vectors]. Arrows connect each label group L to its minimal supersets. In Fig. 2(a), there are* 400 *vectors labeled with A only, shown as $A[400]$. Given a hybrid AKNN query with query label set $L = \{A\}$, Fig. 2(b) shows the relevant vector groups—i.e., those whose label sets are supersets of A—including $\{A, AB, AC, ABC\}$, totaling 1000 vectors that need to index under the query label set $\{A\}$. Fig. 2(c) summarizes the number of vectors needed to index for each query label set. To support all possible queries,* 5400 *vectors must be indexed—2.75× the original dataset size.*

satisfy the query label constraints, removing both the nodes and their connections to neighbors during the search. In contrast, the PostFiltering strategy retains all nodes and connections, but excludes any label-mismatched nodes from being added to the results.

However, the search performance of these two strategies degrades significantly when selectivity is low. Specifically, the PreFiltering strategy suffers from reduced accuracy because removing many mismatched nodes sparsifies the graph, making it difficult to reach the target vectors. While the PostFiltering strategy retains all nodes, it still requires computing distances between the query and numerous mismatched vectors, resulting in inefficient query processing. To address these challenges, existing systems such as Milvus [44], ADB [50], VBASE [58], and CHASE [35] dynamically select between the PreFiltering and PostFiltering strategies based on the query workload and cost estimation. However, the limitations of these search strategies persist. Some heuristic approaches, including NHQ [45] and HQANN [51], propose fusion distances that combine both vector similarity and label matching between the query and database vectors. Nevertheless, these methods require manual tuning of the weight parameters for the vector and label components, and their performance still falls significantly short of the state-of-the-art.

State-of-the-Art. To support label-hybrid AKNN search, ACORN [40] and UNG [5] represent the state-of-the-art. ACORN extends the PreFiltering strategy to address connectivity issues under low selectivity by introducing a parameter γ that increases the graph density—each node has γ times more outgoing edges than in a standard index. This denser graph improves robustness when filtering out mismatched nodes during query processing. However, ACORN ignores label information during index construction, leading to potential result incompleteness.

To address this, UNG leverages label containment during graph construction. It groups base vectors by label sets and builds a subgraph for each group. Subgraphs for label set L are linked to its minimal superset via cross-group edges, ensuring that any vector

Table 1: We compare our approach with existing solutions across 4 dimensions. Search Performance measures the accuracy and efficiency of a method and is validated through experiments. Efficiency Guarantee captures the theoretical time complexity, correctness assurance, and predictability of performance. Index Flexibility reflects the ability to support label-hybrid AKNN search without being tied to a specific index structure. Space Utilization evaluates how effectively the method balances indexing cost and storage efficiency.

Feature	ELI (Our)	NHQ	UNG	ACORN	Filtered
Search Performance	$\star\star\star$	$\star$	$\star^{\uparrow}$	$\star^{\uparrow}$	$\star$
Efficiency Guarantee	✓	×	×	×	×
Index Flexibility	✓	×	×	×	✓
Space Fully Utilize	✓	×	×	×	×

with a superset of L is reachable from group L's entry points—thus guaranteeing completeness. Nevertheless, both ACORN and UNG are limited to graph-based indexes and lack *index flexibility*. Moreover, neither method offers theoretical or practical *search efficiency guarantees*. Experimental results show significant performance degradation as label set size grows, and both approaches fail to *fully utilize resources* under tight space constraints.

Motivation. The challenge with existing solutions stems from the label set associated with vectors in the database S. Let $\mathcal{L}$ denote the query label sets workload. When data cardinality is large, existing methods become inefficient for label-hybrid AKNN search. A naive solution is to build a separate index for every possible query label set in $\mathcal{L}$. Under this approach, a vector with label set $L \subseteq \mathcal{L}$ must be inserted into all subsets of L for indexing. As shown in Fig. 2, a vector with labels $\{A, B, C\}$ must be indexed in 8 groups: $\{\emptyset, A, B, C, AB, AC, BC, ABC\}$—each corresponding to a possible query label set. Here, $\emptyset$ represents queries with no label constraints. This results in significant overhead, as each vector must be inserted into multiple label-set groups. Empirical studies show that when the average label set size is 6–10, this leads to 64×–1024× more index entries compared to the original database S, incurring prohibitive indexing costs and storage overhead.

Our idea. As discussed, indexing all possible query label sets is impractical due to the large number of indices, leading to high index costs. A natural question arises: can we select a subset of $\mathcal{L}$ to index (thus reducing cost) while still ensuring efficient label-hybrid AKNN query processing? This is non-trivial—if a subset $L \subseteq \mathcal{L}$ is not indexed, then queries with label set L cannot be answered effectively. Fortunately, we provide a theoretical analysis of label containment in AKNN search. We show that an index built for a label set L can also answer queries for any superset $L' \supseteq L$. Moreover, we prove that the query latency in this case is bounded by an *elastic factor*—the ratio of the number of vectors matching L to those matching L'. Note that a higher *elastic factor* implies better search efficiency in theory and practice. For example, in Fig. 2 (right), vectors in label group $\{A\}$ overlaps with those of group $\{AB\}$ by 0.5. Thus, the index built on $\{A\}$ can efficiently answer $\{AB\}$. Group $\{B\}$ has even higher overlap with $\{AB\}$, so using its index yields better efficiency. With this in mind, we formalize the index selection problem: can we choose subsets of $\mathcal{L}_q$ to index such that both the space usage and the elastic factor are bounded?

We first prove that this decision version of the index selection problem is NP-complete. We then formulate two optimization variants of the problem: (1) the efficiency-constrained version, which aims to minimize index space under a lower bound on the elastic factor (a measure of search efficiency), and (2) the space-constrained version, which aims to maximize the elastic factor given a fixed space budget. We propose a greedy algorithm for the first variant and extend it to solve the second. To conclude, our methods selectively index label sets to efficiently support label-hybrid AKNN search using high elastic factors. Compared to existing approaches (Table 1), we offer greater flexibility, theoretical guarantees, better search efficiency, and effective space-performance trade-offs.

Contribution. We summarize our main contributions as follows:

Problem Analysis (§ 3). We identify three key limitations in existing label-hybrid AKNN solutions: limited flexibility, no theoretical guarantees on efficiency, and suboptimal performance. To address these, we analyze search performance using the elastic factor, which quantifies how an index on label set L can support queries over its supersets, motivating our novel index selection problem.

Index Selection Problem Formulation (§ 3). We formally define the decision version of the index selection problem: given a query workload $\mathcal{L}$, can we select label sets from its subsets such that the total index space is below a given bound, and the elastic factor exceeds a threshold? We prove this problem is NP-complete, indicating that finding exact solutions is intractable.

Index Selection Problem Solution (§ 4). Building on the decision version, we define two practical optimization variants of the index selection problem: (1) the efficiency-constrained version, which minimizes space given a lower bound on the elastic factor; and (2) the space-constrained version, which maximizes the elastic factor under a space limit. These formulations enable users to balance space and query efficiency. We design a greedy algorithm for the first variant and extend it to solve the second.

Extensive Experiments (§ 5). We evaluate our algorithm on multiple real-world datasets with diverse label distributions. Experimental results show that our approach achieves near-optimal retrieval efficiency with only a 1× increase in space overhead. Moreover, it consistently delivers robust performance on large-scale datasets with extensive label sets, achieving 10×–800× speedups over state-of-the-art baselines.

Due to space limitations, some discussion and experiments are omitted, which can be found in our technical report [56].

2 PRELIMINARY

We formally define the label-hybrid approximate k nearest neighbor search problem in § 2.1. We then review existing solutions in § 2.2, followed by a detailed analysis of the problem in § 2.3.

2.1 Problem Statement

Each entry (x_i, L_i) in the dataset S consists of a d-dimensional vector $x_i \in \mathbb{R}^d$ and an associated label set L_i, which may be empty, and all possible label is organized in Σ. A label-hybrid search is denoted by a query (q, L_q), where q is the query vector and L_q is the query label set. When L_q is specified, we first filter the dataset S to obtain a candidate subset $S(L_q) \subseteq S$, defined as: $S(L_q) = \{(x_i, L_i) \in S \mid L_q \subseteq L_i\}$, i.e., the entries/vectors whose label sets contain all labels in L_q. The label-hybrid k-nearest neighbor search is performed over this candidate subset. Formally,

Definition 2.1 (Label-Hybrid KNN Search). Given a label-hybrid dataset S and a query (q, L_q), the label-hybrid KNN search aims to return a set $S' \subseteq S(L_q)$ of k entries such that for every $(x_i, L_i) \in S'$ and every $(x_j, L_j) \in S(L_q)$, it holds that $\delta(x_i, q) \leq \delta(x_j, q)$.

However, exact nearest neighbor search suffers from the *curse of dimensionality* [26], causing traditional methods [3, 4, 39] that perform well in low-dimensional spaces to degrade significantly in high-dimensional settings. As a result, approximate nearest neighbor (ANN) search has been extensively studied [1, 7, 10, 11, 14–18, 28, 30, 33, 37, 47, 49, 53, 55, 59] for its ability to significantly improve efficiency at the cost of some accuracy. This challenge also arises in the context of label-hybrid search. Therefore, we focus on the label-hybrid approximate k nearest neighbor (AKNN) problem. To evaluate the result quality, we use *recall* as the metric, defined as recall $= |\hat{S} \cap S'|/|S'|$, where S' is the exact result set and $\hat{S}$ is the approximate result. An effective label-hybrid AKNN solution must balance high search efficiency with high recall.

Remark. The label-hybrid AKNN search can be viewed as a special case of the filtered nearest neighbor search problem [19], where filtering is based on label set inclusion (i.e., requiring the label set L_i of a base vector in the database to include the query label set L_q). This paper focuses on set inclusion, while cases involving set intersection ($L_q \cap L_i \neq \emptyset$) or set equality ($L_q = L_i$) can be addressed via straightforward transformations and partition-based indexing. We provide a detailed analysis in § 3.

2.2 Existing Solutions

To address the label-hybrid AKNN search problem, existing approaches typically combine different types of AKNN indexes with filter-based search strategies. Among these, graph-based indexes [11, 24, 28, 34, 37, 41] are widely adopted due to their state-of-the-art search efficiency. In such indices, base vectors in the dataset S are treated as nodes in a graph, where each node is carefully connected to its nearby vectors, enabling efficient navigation.

Graph Index. The graph search begins from a designated entry node and iteratively explores neighbors that are increasingly closer to the query until the top results are identified. A critical component of this design is the edge occlusion strategy, which ensures that the search progresses toward the query while keeping the graph sparse. Notably, this strategy allows the node degree to be bounded by a constant [11, 27].

To support AKNN search—which requires retrieving multiple results—beam search is commonly employed. It maintains the current top-m closest candidates during traversal, where m denotes the beam width. Under ideal indexing conditions (i.e., high index quality and when the query is within the dataset) [11, 27], only one additional step beyond retrieving the k-th neighbor is needed to reach the $(k+1)$-th, and the overall search complexity is logarithmic in the size of the dataset.

Filter-Based Search. On top of the graph index, we introduce two filter-based search strategies—PreFiltering and PostFiltering—to support label-hybrid AKNN search. Both strategies maintain the

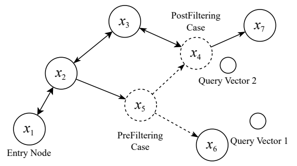


Figure 3: The Example of Graph Search

original index structure and require only label-based filtering during graph traversal. (1) PreFiltering strategy: Given a query (q, L_q), PreFiltering prunes unmatched nodes and their neighbors during traversal. As shown in Fig. 3, suppose nodes x_4 and x_5 do not satisfy the label constraint in query 1. In this case, PreFiltering removes the outgoing edges of x_5, thereby making the true nearest neighbor x_6 unreachable from the entry node x_1. (2) PostFiltering strategy: This method allows traversal through unmatched nodes but excludes them from the result set. For instance, when processing query 2 in Fig. 3, PostFiltering visits nodes x_1, x_2, x_3, x_4, and x_7. Although x_4 is unmatched, its outgoing edge is still used for routing, enabling the algorithm to eventually reach and return x_7.

For top-k queries, PostFiltering continues traversing until it accumulates k matching candidates, typically by visiting the $k + 1$ nearest neighbors. However, its time complexity depends on query selectivity: if most nodes are filtered out, the algorithm may need to examine up to $O(N)$ nodes to retrieve k results. PreFiltering faces a similar challenge—under low selectivity, it may fail to reach any valid neighbors from the entry node, making its correctness and efficiency difficult to guarantee.

Remark. Using filtered-based search strategies for label-hybrid AKNN queries often leads to suboptimal performance when the selectivity is low. While techniques such as query planning can choose between the PreFiltering and PostFiltering strategies based on query workload characteristics, the filter-based approach still suffers from a significant performance gap compared to the optimal solution that indexes only the filtered vectors.

State-of-the-art. To address the performance limitations of filter-based methods, specialized algorithms have been developed for label-hybrid AKNN search. NHQ incorporates label information into the distance function by defining a fused distance metric that combines vector similarity and label similarity. This approach, however, requires manual tuning of the relative weights assigned to vectors and labels. Although an empirical method for weight setting is provided, NHQ still lags behind more recent methods such as ACORN and UNG in overall performance. Furthermore, the modified distance metric complicates the analysis of both time complexity and algorithmic soundness.

Recall that the main issue with PreFiltering is that eliminating unmatched nodes sparsifies a graph, potentially disrupting connectivity. The recent method ACORN addresses this by constructing a denser graph index using a compressed neighbor list. For label-hybrid AKNN search, PreFiltering operates on a subgraph consisting of matched points. A denser graph can alleviate the reachability problem caused by PreFiltering, but cannot fully eliminate it.

To overcome this limitation, UNG leverages the containment relationship between label sets: it connects each node group (under a label set L) to the nodes of its smallest superset via cross-group edges. This design ensures that the search is restricted to matched nodes while preserving graph connectivity. Yet, both ACORN and UNG rely on the navigability of the graph structure, making them incompatible with non-graph-based vector indexes. Furthermore, due to their heuristic designs, neither ACORN nor UNG can guarantee search efficiency in terms of time complexity, and their empirical performance still lags behind that of the optimal approach.

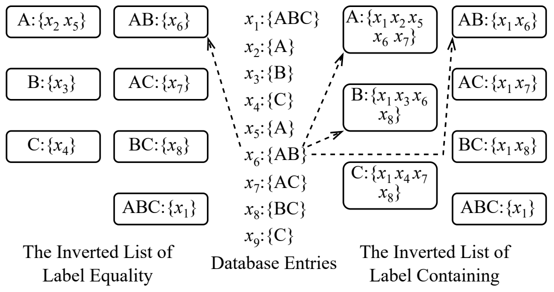


Figure 4: The Inverted List of Label Equality and Label Containing

2.3 Problem Analysis

To conclude, existing solutions fall short in effectively handling label-hybrid AKNN search. In response, we present an in-depth analysis of the core challenges, guided by the perspective of an optimal search strategy.

Challenges. We aim to achieve optimal search performance using existing AKNN indexing algorithms. A natural approach is to pre-build indexes over base vectors that match the query label set, assuming access to historical query information.

For the label-containing query, achieving optimal indexing performance incurs significantly higher costs. As illustrated on the right side of Fig. 4, to build the index, an entry such as $(x_6, \{AB\})$ in the database S must be inserted into multiple inverted lists—specifically for label sets $\{A\}$, $\{B\}$, $\{AB\}$, and an additional list for label-free AKNN queries (i.e., the label set is $\emptyset$). This is necessary because, according to the problem definition, a base vector matches a query if its label set is a superset of the query label set. Therefore, an entry (x_i, L_i) must be indexed in all subsets of L_i, resulting in $2^{|L_i|}$ insertions if all such label-hybrid queries need to be supported. For instance, the entry $(x_1, \{ABC\})$ is inserted into every subset-based index shown in Fig.4 (right).

Thus, the optimal indexing method incurs substantial overhead in both indexing time and storage space, particularly when the average label set size is large. Although it is possible to embed less significant labels into the vector representation and not all label combinations may be queried in practice, an average label set size of 10 can already lead to an index that is over 1000× larger than a label-free baseline—severely limiting its scalability and usability.

Motivation. Building on the previous analysis, the primary obstacle to achieving the optimal approach lies in the exponential index space and construction time with respect to the average label set size. A natural solution is to reduce the amount of indexed data as much as possible while maintaining query efficiency. This is feasible because, in label-containing queries—which are more general and form the focus of this paper—the index exhibits a containment relationship among label sets.

For instance, as shown on the right of Fig. 4, the inverted list for the label set $\{B\}$ subsumes the data of its supersets, such as $\{AB\}$ and $\{BC\}$. By leveraging the index of $\{B\}$ in combination with index-free methods (e.g., graph index with filter-based search), we can answer label-containing queries for both $\{AB\}$ and $\{BC\}$. This insight motivates our strategy to selectively build indexes for representative label groups, allowing shared indexes to serve multiple label-containing queries. As a result, we significantly reduce both index space and construction cost. The following sections detail how this strategy enables efficient search through index selection.

3 ELASTIC INDEX SELECTION PROBLEM

This section formally defines the index selection problem, aiming to overcome the exhaustive cost in indexing space and time caused by exponential label combinations. The key challenge lies in how to quantify query efficiency. To address this, we first introduce the elastic factor, which captures how an index built on a label set can be reused for its supersets. We then formally define the index selection problem as a decision problem: determining whether it is possible to select a subset of label sets for indexing such that both space usage and query efficiency remain within given bounds. Finally, we prove that the problem is computationally hard.

3.1 Elastic Factor for Index-Sharing

We begin by examining the feasibility of index sharing. Specifically, an index built on a label set L_2 can serve as a shared (or substitute) index for another label set L_1 only if it contains all base vectors associated with L_1. This condition is satisfied when $L_2 \subseteq L_1$, as every base vector matching L_1 will also match its subset L_2. For instance, the entry group labeled with $\{A\}$ includes all entries from its superset group $\{AB\}$, since any vector labeled $\{AB\}$ also contains label $\{A\}$.Consequently, queries can leverage indexes built on supersets of their label sets.

Elastic Factor. We next analyze the query efficiency enabled by shared indexes. Instead of constructing indexes for all possible label sets, we selectively build a subset of indexes $\mathbb{I} = \{I_1, \ldots, I_m\}$. Note that each index is built over a subset of base vectors in the dataset S under a specific label set. Without ambiguity, we sometimes call the index the base vectors that it is constructed. The efficiency of querying with shared indexes depends on the selectivity of the query, which is determined by the query label set L_q. To quantify this, we introduce the concept of the elastic factor, which captures the maximum number of base vectors that can match the query label set L_q under the selected index set.

Definition 3.1 (Elastic Factor). Given a label-hybrid dataset S, the query (q, L_q) and a set index $\mathbb{I} = \{I_1, ..., I_m\}$, each index is a subset of S. The elastic factor of a index set with $I_i \in \mathbb{I}$ is defined as:

$$e(S(L_q), \mathbb{I}) = \max_{S(L_q) \subseteq I_i} \left(\frac{|S(L_q)|}{|I_i|} \right)$$

Connection to Query Efficiency. In extreme cases, one can perform a filter-based search (e.g., PostFiltering) over an index built on all entries—corresponding to the label set being the empty set, which is a subset of any label set. However, this approach is inefficient for queries with non-empty label sets L_q, as it may traverse a large number of unmatched nodes in the graph index. This inefficiency arises because the elastic factor, when using only the index built on the empty set, tends to be very low for queries with specific label constraints. What happens when the elastic factor is guaranteed to be no smaller than a constant C?

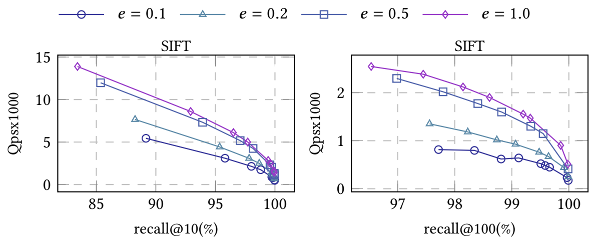


Figure 5: The test of verifying elastic factor and query efficiency. We randomly generate labeled data and queries. We built HNSW indexes for the original data and divided the queries into four groups according to the elastic factors (selectivity): 0.1, 0.2, 0.5, and 1. The case of $e = 1$ can be regarded as the optimal approach.

To answer this question, we first present the theoretical result for KNN search. Theoretically, any index supporting top-k nearest neighbor search can perform incremental $(k+1)$ searches to accumulate k unfiltered results for filtered search. Thus, the performance depends on the expected number of $(k+1)$ searches needed to retrieve k valid entries. When using an index built over the entire dataset S (as in PostFiltering search), the expected number of steps $\mathbb{E}(r)$ can reach $O(N)$ under low query selectivity, explaining its inefficiency. However, if the query is answered using an index with a constant elastic factor c, the expected number of required $(k+1)$ searches can be bounded by $O(k/c)$.

This holds because, in graph-based indexes, retrieving the $(k+1)$-th nearest neighbor typically requires only one additional step under ideal conditions (e.g., slow growth, query present in the dataset). Therefore, if a label-hybrid AKNN search shares an index with a minimum elastic factor of c, the overall time complexity remains asymptotically unchanged—assuming independence between labels and vectors—with only an additional factor of k/c. We formalize this in Lemma 3.2 [54].

LEMMA 3.2. *Given a label-hybrid dataset S, a query (q, L_q), and an index set $\mathbb{I}$, let $O(C)$ denote the top-1 search time complexity of the graph index. If the elastic factor satisfies $e(S(L_q), \mathbb{I}) = c$ for some constant c, then the expected time complexity of the label-hybrid AKNN search using the graph index is $O(C + k/c)$.*

In the above lemma, we denote by C the time cost of retrieving the top-1 result using a graph index. This is because different graph indexes have varying search complexities, we then abstract this cost as C, which typically grows logarithmically with the dataset size $N = |S|$. The above time complexity guarantee holds only when the graph index is constructed with $O(N^3)$ time [11, 27], which is impractical for large-scale vector search. In practice, graph indexes adopt heuristic methods to accelerate construction [11, 24, 34, 37], such as considering only partial neighbors for edge occlusion, leading to approximate results. Moreover, the theory in [11] assumes the query is in the database, while in practice it may lie anywhere.

Empirical Studies. We evaluate the practical efficiency of PostFiltering search under varying elastic factors on the SIFT dataset. In the original PostFiltering design, the elastic factor is fixed, as it operates on the label group containing all base vectors (i.e., an empty label set). We vary the elastic factor and report the results in Fig. 5. As shown, higher elastic factors lead to better efficiency, aligning with the previously analyzed lower time complexity. Notably, when $e(\cdot,\cdot) = 1$, the search achieves optimal efficiency, equivalent to indexing only the matched data of the query label set.

Also, we observe that the search efficiency degrades sublinearly with the elastic factor. When the elastic factor is reduced to $\frac{1}{10}$ of the optimal, the efficiency drops by only 2× at 98% recall with k=10. This is because the increased time complexity is bounded by a factor relative to k, while the AKNN search still operates in $\log N$ time. The performance gap becomes more pronounced for larger k values, with a 3× slowdown at k=100 and 98% recall. Yet, in most applications, k is small relative to N. Thus, as long as the elastic factor remains a constant, the overall time cost remains bounded.

3.2 Problem Definition

We have analyzed the impact of the elastic factor on search efficiency. The elastic factor is introduced to reduce indexing space by selectively indexing only a subset of label sets, while still ensuring efficient query performance. Importantly, when using graph indexes, the query cost does not scale linearly with the size of the database. Instead, we use the elastic factor to model query efficiency: the cost of a query is determined by its associated elastic factor. Based on this observation, we aim to select a set of indexes such that the elastic factor for any query exceeds a constant lower bound c. This guarantees that the top-k search algorithm scales at most with k/c in theory, enabling control over c to balance efficiency and index size. Motivated by this, we formally define the Elastic Index Selection (EIS) problem.

*Definition 3.3 (Fixed Efficiency Index Selection (*EIS*)).*

Input Given a label-hybrid dataset S, a query workload with label sets $\mathcal{L} = \{L_1, \ldots, L_n\}$, and an index set $\mathbb{I} = \{I_1, \ldots, I_n\}$ where each index $I_i = S(L_i)$ corresponds to the selected data for query L_i, and each index has a cost $|I_i|$. Let τ be a non-negative real number.

Output A subset $\mathbb{I}' \subseteq \mathbb{I}$ such that $e(\mathbb{I}', S(L_i)) > c$ for all $L_i \in \mathcal{L}q$ and total cost $\sum I \in \mathbb{I}'|I| < \tau$.

The EIS problem is a *decision* problem to check whether we can select a subset of the necessary indexes such that (1) the total space usage is bounded by a user-defined parameter τ, and (2) the elastic factor—which governs query efficiency—is no less than a specified threshold c. We illustrate this problem with the example shown in Fig. 6. In the following section, we discuss the *optimization* version of the problem, where the goal is either to minimize the space cost while ensuring the elastic factor meets a given threshold or to maximize the elastic factor under a space constraint.

Remark on Query Workload. In the above problem definition, we assume access to a query workload represented by a collection of label sets $\mathcal{L}_q = \{L_1, \ldots, L_n\}$. This assumption is justified for two key reasons. (1) It generalizes the setting by allowing enumeration of all possible subsets of the label alphabet Σ, which contains all labels in the problem; these subsets can be stored in $\mathcal{L}_q$ to support comprehensive analysis. (2) It enables a workload-driven approach, where $\mathcal{L}$ reflects historical query data, allowing us to tailor index selection and optimization strategies to realistic query patterns.

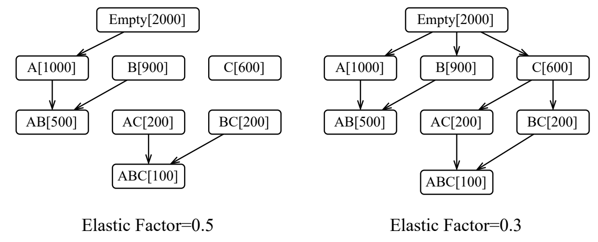


Figure 6: The cover relationship of the query index.

Example 3. *Fig. 6 shows index selection outcomes under different elastic factor settings, specifically $e = 0.5$ and $e = 0.3$. For $e = 0.5$, the top index (constructed on the empty label set) can answer queries with $L_q = \{A\}$ and $L_q = \emptyset$, since the overlap between group $\{A\}$ and the top index is at least 0.5. However, it cannot support the query $L_q = \{B\}$, as the overlap is only 0.45. Under a more relaxed threshold of $e = 0.3$, the top index can additionally support queries with $L_q = \{B\}$ and $L_q = \{C\}$. Note that it still cannot answer the query $L_q = \{AB\}$, whose overlap is only 0.25—this would only be feasible if $e < 0.25$. When $e = 0$, the top index can serve all possible queries, regardless of label constraints.*

Remark on Query Label Sets. Following the above assumption, when considering only the label component of the query, the number of possible label sets is $2^{|\mathcal{L}|}$ in the worst case. For each label set, we can identify the matched base vectors in the database and build an index accordingly. In practice, this number is often smaller due to the orthogonality among certain labels. We also assume that the query workload includes a label set $\emptyset$, meaning the query imposes no constraints on labels and should apply to all base vectors. In this case, we exclude the cost of the top-level index to simplify the problem (i.e., $|I_{\text{top}}| = 0$).

Remark on Index Size. The cost of each index corresponds to the space it occupies under a given label set. When employing a graph-based index, this cost can be approximated by the number of vectors it contains, as the node degree is typically bounded by a constant [11, 27]. In practice, each node maintains M edges to support fast memory access, where M is a user-defined parameter [36]. Thus, the overall space usage of the index set can be estimated by multiplying the total number of vectors by M.

3.3 Problem Hardness

We proved that the EIS problem is NP-complete in Theorem 3.4, underscoring the computational challenges inherent in solving it.

Theorem 3.4. *The* EIS *problem is NP-complete.*

Proof. We first prove the NP-hardness of the EIS problem via a reduction from the NP-complete problem 3-Set Cover (3-SC) [8, 31].

Input A universal set $\mathbb{U}$ containing p elements, denoted as $\mathbb{U} = \{u_1, u_2, \cdots, u_p\}$, a set $\mathbb{S}$ containing l subsets, denoted as $\mathbb{S} = \{s_1, s_2, \cdots, s_l\}$, where each $s_i \in \mathbb{S}$ contains up to 3 elements from $\mathbb{U}$. A non-negative number k.

Output A non-empty subset from $\mathbb{S}$, denoted as $\mathbb{A}$ whose union is still $\mathbb{U}$ and the size is less than k.

Given a 3-SC instance defined above, we could generate an instance of the EIS problem in Fig. 7 where the arrow denotes the cover relationship. As illustrated in Fig. 7 (a), each element s_i from $\mathbb{S}$ and u_i from $\mathbb{U}$ in 3-SC instance are mapped to an index in $\mathbb{I}$ except the top and bottom one. The top index is built with all base vectors, and the bottom index is built with base vectors that contain all possible labels. In this case, the query label set is limited to label combinations that appear in base S. From Fig. 7 (b), each label set of s_i index is set as a single element S_i. If $u_i \in s_i$, we add S_i to the label set of u_i index, and the init label set is U_i. This can ensure the s_i index can cover u_i. Additionally, we add a duplicate index u'_i that has the same label set as u_i except the init label U_i, which is set as U'_i. We add the duplicate index to ensure that the cost of using s_i to cover u_i is lower than the cost of selecting u_i alone. Next, we design the cost (size) of each index and the elastic factor bound c. The cost of u_i and u'_i is 11, since the index contains one entry with the corresponding label set and 10 entries from the bottom of Fig. 7. For example, the query index u_1 contains 1 entry with $\{S_1S_2U_1\}$ and 10 entries with all labels for containing query label set $\{S_1S_2U_1\}$. The cost of s_i index is set to 20, where up to 3 u_i is covered by s_i, and the number of entries with label S_i is set to 10-2$\times|s_i|$. For instance, the s_2 index contains 4 entries with label S_2 and 6 entries from $u_1, u_2, u_3, u'_1, u'_2, u'_3$, and 10 from bottom. The cost of s_i is equal to adjusting the number of S_i label entries. Then, the elastic factor can be set to $20/N < c < 0.5$ such that the top index can cover the s_i index but can not cover u_i, and the s_i index can cover u_i if $u_i \in s_i$.

Next, we show that solving the EIS problem is equivalent to solving the 3-SC instance.

<u>*Solution to 3*-SC $\Rightarrow$ *Solution to* EIS.</u> For EIS, the top index can cover index s_i, and the cost is excluded as we discussed before. Since each s_i has the same cost (cost=20), we set $k = \lfloor \tau/20 \rfloor$. Assuming 3-SC can be solved, then we can determine whether a subset $\mathbb{A}$ of $\mathbb{S}$ with up to k elements is equal to the universal set $\mathbb{U}$. If the solution $\mathbb{A}$ for 3-SC exists, we select $s_i \in \mathbb{A}$ to cover all u_i in Fig. 7 and the cost is $k \times 20 \leq \tau$. Any selected index can cover the bottom index. Then, all $L_i \in \mathcal{L}_q$ are covered with an elastic factor greater than c.

<u>*Solution to* EIS $\Rightarrow$ *Solution to 3*-SC.</u> For a 3-SC instance with parameter k. We can still set $\tau = k \times 20$. If the solution $\mathbb{I}'$ of EIS exists, we can get a subset of $\mathbb{I}$ that covers all $I : u_i$ indices. As we discussed, the cost of index u_i is greater than any s_i containing u_i, and any s_i can cover the bottom index. We transfer the u_i in solution $\mathbb{I}'$ to any s_i that contain u_i. This operation will only reduce the cost without affecting the correctness of the solution. Then we can get the solution $\mathbb{A}$ for 3-SC by select s_i in $\mathbb{I}'$. Since the cost of each element in $\mathbb{I}'$ is 20 and the threshold τ is $k \times 20$, the number of elements can only be less than or equal to k. This indicates that $\mathbb{A}'$ is the solution to 3-SC.

Given a solution to the EIS problem, we can verify in polynomial time whether it satisfies the constraints on space and elastic factor. Therefore, EIS is in NP. Since we have already shown that EIS is NP-hard, it follows that EIS is NP-complete. □

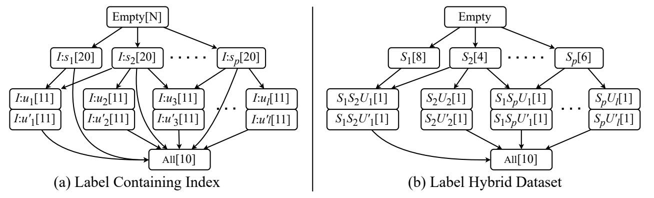


Figure 7: The Proof of NP-complete.

Note that the EIS problem involves two constraints: a space constraint and an elastic factor constraint. In the following, we study two optimization variants of EIS. The first focuses on minimizing the total space under a given lower bound on the elastic factor, referred to as the **efficiency-constrained variant**. The second aims to maximize the elastic factor while satisfying a space constraint, referred to as the **space-constrained variant**. Due to the NP-completeness of EIS, both optimization variants are NP-hard. Therefore, exact solutions are computationally intractable in practice. We propose to address these problems using greedy heuristics.

4 PROBLEM SOLUTION

In this section, we first design a greedy algorithm to solve the efficiency-constrained variant of the EIS problem, and then extend it to address the space-constrained variant.

4.1 Method For Efficiency-Constrained Variant

This optimization version of the EIS problem resembles Definition 3.3, but aims to compute the following output:

Output A subset $\mathbb{I}' \subseteq \mathbb{I}$ such that $e(\mathbb{I}', S(L_i)) > c$ for all $L_i \in \mathcal{L}q$, and the total cost $\sum I \in \mathbb{I}'|I|$ is minimized.

Since the EIS problem is NP-hard, we resort to a greedy algorithm. The basic idea is as follows: We consider a collection of indexes, each associated with a space cost, measured by the number of matched vectors it contains. For the EIS problem, we define a benefit function for each index and iteratively select the index with the highest benefit, continuing until the elastic factor constraint is met. Importantly, adding more indexes to the final solution can decrease the overall elastic factor. Therefore, index selection must be performed carefully to avoid violating the constraint.

One tricky aspect is that we always include the top index (corresponding to the empty label set) in the selected set. As discussed in Section 3.2, one reason for this is to ensure that each query has at least one index covering all its matched vectors. Therefore, the answer set $\mathbb{I}'$ initially includes the top index. Then, we denote the benefit of adding a candidate index $I' \in \mathbb{I} \backslash \mathbb{I}'$ to the current selection $\mathbb{I}'$ as $B(I', \mathbb{I}')$, and is defined as follows:

Definition 4.1. Let $\mathbb{C} \subset \mathbb{I}$ be the index set covered by the selected index set $\mathbb{I}'$ under elastic factor c where:

$$\mathbb{C} = \{I_i \in \mathbb{I} \mid \exists I_j \in \mathbb{I}' \ s.t.(I_i \subseteq I_j) \land (|I_i|/|I_j| > c)\}.$$

Let $\mathbb{C}'$ denote the set of index covered by I_i where:

$$\mathbb{C}' = \{I_i \in \mathbb{I} \setminus \mathbb{C} \mid (I_i \subseteq I') \land (|I_i|/|I'| > c)\}.$$

The benefit $B(I', \mathbb{I}')$ is defined as:

$$B(I', \mathbb{I}') = \sum_{I_i \in \mathbb{C}'} |I_i|/|I'|$$

```
Algorithm 1: Greedy Algorithm
  Input: The Index Set $\mathbb{I}$, The Elastic Factor $c$, The Threshold $\tau$
  Output: The Selected Index Set $\mathbb{I}'$
1 $\mathbb{I}' \leftarrow \{\text{Top Index}\}$;
2 while $\sum_{I_i \in \mathbb{I}'} |I_i| \leq \tau$ do
3     $I \leftarrow I_i \in \mathbb{I} \setminus \mathbb{I}'$ such that $B(I_i, \mathbb{I}')$ is maximum;
4     $\mathbb{I}' \leftarrow \mathbb{I}' \cup \{I\}$;
5     if For any $I_i \in \mathbb{I}$ that $e(I_i, \mathbb{I}') > c$) then
6         return $\mathbb{I}'$;                       // Greedy Result
7 return No Solution;
```

To illustrate, consider the case where we only account for the number of indices. The benefit of selecting an index is measured by how many other indices it can cover. The cost of an index is defined as the number of vectors it contains. Thus, the benefit is calculated as the total cost of the covered indices normalized by the cost of the selected index.

Example 4.2. We provide a running example in Fig.8. Fig.8(a) shows the organization of vectors within each label group, where three vectors have no labels and three have the label set $\{ABC\}$, etc. Fig. 8(b) presents the number of vectors considered by each possible query. For instance, for a query with label set $L_q = \{A\}$, all vectors labeled with $\{A\}$, $\{AB\}$, $\{AC\}$, or $\{ABC\}$ match, resulting in a total of 10 vectors. Building an index I_2 on these 10 vectors not only supports the query $L_q = \{A\}$, but also supports queries like $L_q = \{AB\}$ or $L_q = \{A\}$, as it includes all relevant base vectors.

Fig. 8(c) illustrates index sharing under an elastic factor of 0.3. Index I_2 can answer the query $L_q = \{ABC\}$ because the overlap ratio is $3/10 = 0.3$. In contrast, index I_1 cannot support this query since its overlap ratio of $3/17$ is below the threshold. Based on the dataset in Fig.8, we evaluate the benefit of different index selections. Table2 reports the benefit-to-cost ratio for each candidate index.

Algorithm. Next, we summarize the greedy strategy in Algorithm 1. The procedure is self-explanatory: we first add the top index to the answer set $\mathbb{I}'$ (Line 1). Then, at each iteration, we select the next index to add, where each candidate corresponds to a label set in the query workload. The selection is based on the benefit score of each candidate index (Lines 3–4). The process terminates either when all label sets are covered (indicating a valid solution) or when the space constraint is violated (Line 2)

Example 4.3. Continuing the previous example, recall that the top index I_1 must always be selected, so we choose it in the first round regardless of its unit benefit. Selecting I_1 covers the query vectors corresponding to indexes I_2, I_3, I_4, I_6, and itself, totaling $17 + 10 + 7 + 9 + 6 = 49$ vectors at a cost of 17, yielding a unit benefit of $49/17 \approx 2.88$. After selecting I_1, the benefit of the remaining candidate indexes changes. For instance, initially, I_2 can cover I_2, I_5, I_6, I_8, totaling 23 vectors with a cost of 10, resulting in a unit benefit of 2.3. However, in the second round, since I_1 already covers I_2 and I_6, I_2 can now only cover I_5 and I_8, providing a benefit of just 7 vectors at the same cost of 10, reducing its unit benefit to 0.7. Our greedy algorithm continues by selecting the index with the highest updated benefit, as shown in Fig. 8(d). As a result, I_5 is selected in the second round. At this point, I_1 and I_5 cover all queries except for I_7. In the third round, I_7 is selected to complete the coverage, incurring a cost of 5. The total cost of the greedy solution is $17 + 4 + 5 = 26$, but this is not optimal. As shown in Fig. 8(e), the optimal solution instead selects I_3 in the second round. Although I_3 only covers I_5, I_7, I_8 with a unit benefit of 1.71—lower than I_5 due to overlap with I_1—it results in complete coverage with just two indexes: I_1 and I_3. The total cost is $17 + 7 = 24$, outperforming the greedy approach.

Table 2: The Benefits per-Unit of Possible Choices.

	Init Round	Second Round	Third Round
I_1	49 / 17 = 2.88		
I_2	23 / 10 = 2.30	7 / 10 = 0.70	0 / 10 = 0.00
I_3	19 / 7 = 2.71	12 / 7 = 1.71	5 / 7 = 0.71
I_4	23 / 9 = 2.55	14 / 9 = 1.55	5 / 9 = 0.55
I_5	7 / 4 = 1.75	7 / 4 = 1.75	
I_6	9 / 6 = 1.50	3 / 6 = 0.50	0 / 6 = 0.00
I_7	8 / 5 = 1.60	8 / 5 = 1.60	5 / 5 = 1.00
I_8	3 / 3 = 1.00	3 / 3 = 1.00	0 / 3 = 0.00

Time Cost Analysis. The time spent on index selection is also included in the overall index construction time. Empirically, for datasets with power-law distributed labels, index selection contributes only a small fraction of the total construction time. However, in theory, when query label sets are not restricted to the labels present in the base dataset, each data entry may need to be inserted into up to $2^{|L_i|}$ indexes. Consequently, computing the sizes of these indexes incurs a time complexity of $O(\sum 2^{|L_i|})$. While this overhead is negligible when the average label set size is small, it may affect scalability for large or dense label sets.

To scale to large datasets, we adopt a similar approach to [22] by estimating the approximate size of each index using sampling techniques or more advanced cardinality estimation models [23]. Once the sizes of all indices are determined, the benefit of each candidate selection can be efficiently computed. We maintain a max-heap to track the index with the highest benefit at each iteration. Since selecting an index may impact the benefits of up to $2^{|L_i|}$ related indices, we verify if the top entry in the heap has become stale. If so, we update its benefit and reinsert it into the heap.

4.2 Method For Space-Constrained Variant

In the previous section, we addressed the efficiency-Constrained index selection problem. A natural extension is to consider how to select indexes that maximize query efficiency while satisfying a space constraint. This optimization variant of the EIS problem closely follows Definition 3.3, but focuses on computing the following output:

Output A subset $\mathbb{I}' \subseteq \mathbb{I}$ such that the total cost $\sum_{I \in \mathbb{I}'} |I| < \tau$, and the minimum elastic factor $\min(e(\mathbb{I}', S(L_i)))$ over all $L_i \in \mathcal{L}_q$ is maximized.

Monotonicity of Elastic Factor. We observe that the elastic factor bound c exhibits a monotonic property. Specifically, if an index selection $\mathbb{I}'$ satisfies an elastic factor of 0.5 for a given query workload $\mathcal{L}_q$, it also satisfies any bound $c' < 0.5$. Leveraging this property, we reduce the Space-Constrained EIS problem to the Efficiency-Constrained EIS problem. In particular, we employ binary search

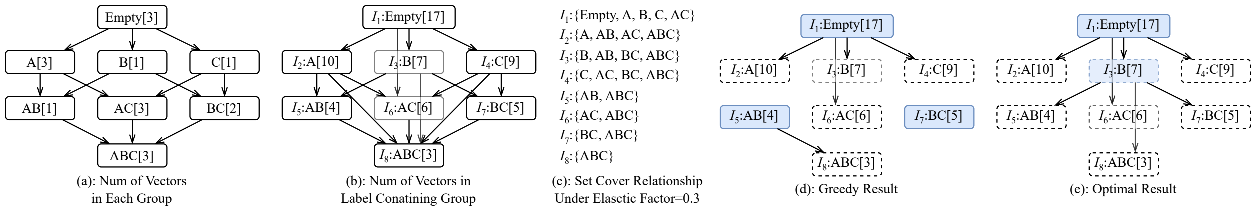


Figure 8: The Running Example.

Table 3: The Statistics of Datasets

Dataset	Dimension	Size	Query Size	Type	Label
SIFT	128	1,000,000	1000	Image	Synthetic
MSMARCO	1024	1,000,000	1000	Text	Synthetic
PAPER	200	1,000,000	1000	Text	Real
LAION	512	1,182,243	1000	Image	Real
TripClick	768	1,020,825	1000	Text	Real
OpenAI-1536	1536	999,000	1000	Text	Real
DEEP	96	100,000,000	1000	Image	Synthetic

on the Efficiency-Constrained EIS problem to determine whether a solution $\mathbb{I}'$ exists such that the elastic factor exceeds c while the total cost remains below a specified threshold τ.

Algorithm. We can thus reuse the greedy selection method developed for the Efficiency-Constrained EIS problem to solve this variant. Specifically, we apply a binary search to identify the optimal elastic factor bound c. During the search, we update the result whenever the greedy solution satisfies the cost constraint τ while achieving a higher elastic factor. This approach requires only $O(\log)$ invocations of the greedy algorithm. In practice, the overhead introduced by combining binary search with the greedy method is minimal—typically less than 1% of the total index construction time, amounting to only 1–2 seconds even for large $\mathcal{L}$.

5 EXPERIMENT

5.1 Experiment Settings

Datasets. We utilize seven public vector datasets, which are widely used in benchmarking and evaluation(SIFT, PAPER)[1]. Some of them are generated by state-of-the-art AI embedding models (MSMARCO[2] OpenAI-1536[3] TripClick[4]). For the Label part, except for the PAPER, LAION, TripClicka, and OpenAI-1536 datasets, which come with real labels, we use the method from previous works [5, 19] to generate label data with different distributions for other datasets. For the PAPER dataset with more than 1M data, we sample 1 million entries. Moreover, we also sampled 100M data from the deep1B[5] dataset to verify the scalability of our algorithms.

Metrics. The overall evaluation metrics involve both search efficiency and accuracy. For the search efficiency metric, we use query per second (QPS), which indicates the number of queries processed by the algorithm per second, as it is most commonly used in benchmarks. For the search accuracy, we use *recall* defined in § 2.1 as the metric to align with the baselines [5, 40]. All metrics used in the experiment are reported as averages.

Label Distribution. In real scenarios, keywords, tags, and labels often approximate the power law distribution. Previous work adopts the power law distribution, the Zipf distribution, to generate labels [5, 19]. We use its original code to generate label data for base vectors with varying the number of possible labels in the alphabet Σ where $|\Sigma|$ =8, 12, 24, and 32. We also consider real datasets and other label distributions, such as Uniform and Poisson, in subsequent experiments. For TripClick, LAION, and OpenAI-1536 datasets, we derive all vectors and categorical labels using the approach outlined in [40].

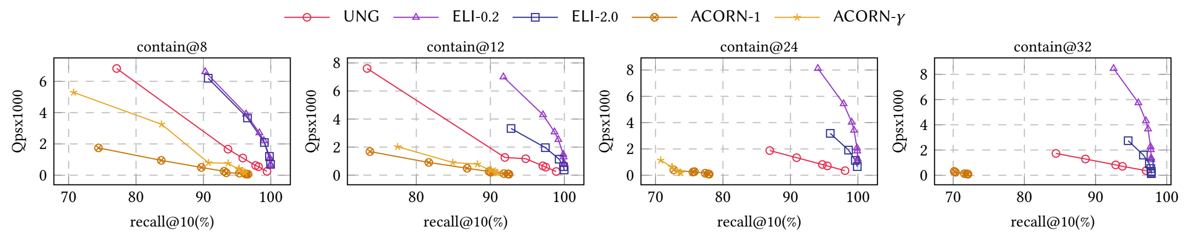
Algorithms. The algorithms compared in our study are as follows:
- ELI-0.2: Our proposed method with fixed search efficiency: elastic factor bound set to 0.2.
- ELI-2.0: Our proposed method with fixed index space: use at most double the original index space.
- UNG: Unified navigating graph approach based on label navigation graph [5].
- ACORN-1: ANN constraint-optimized retrieval network with low construction overhead [40].
- ACORN-γ: ANN constraint-optimized retrieval network for high-efficiency search [40].

Implementation Details. All code was implemented in C++ and compiled using GCC version 9.4.0 with -Ofast optimization. The experiments were conducted on a workstation with Intel(R) Xeon(R) Platinum 8352V CPUs @ 2.10GHz, 512GB of memory. We utilized multi-threading (144 threads) for index construction and a single thread for search evaluation. We use HNSW [37] as the modular index with parameters M=16 and *efconstruct*=200. For ELI-0.2(<1.0), we use the index selection method to achieve a fixed elastic factor of 0.2. For ELI-2.0(>1.0), we use the fixed space method of at most double the origin index space to achieve the maximum efficiency. For other baselines such as ACORN and UNG, we use the default parameters in their papers, i.e., α=1.2 L=100 for UNG and γ = 80, M = 128, M_B = 128 with ACORN-γ for LAION and TripClick datasets and γ = 12, M = 32, M_B = 64 for the other datasets.

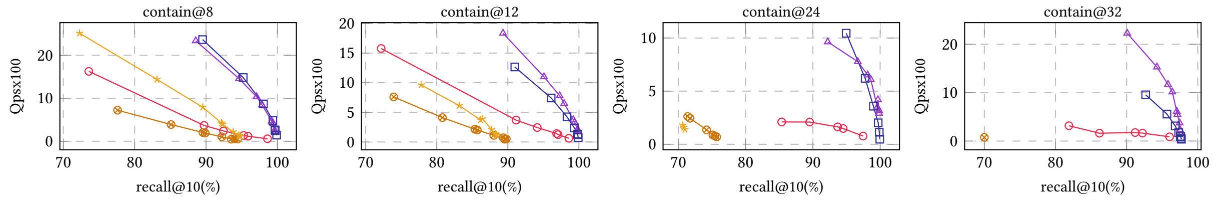
5.2 Experiment Results

Exp-1: Query Efficiency Performance. We start our evaluation from the query efficiency of different algorithms. We compare UNG, ACORN-1. ACORN-γ and our method ELI-0.2 and ELI-2.0 under different label set size settings for label containing queries in Fig. 9(top right is better). From the result in Fig. 9, our algorithm achieves the best search efficiency and accuracy tradeoff. At 95% recall and

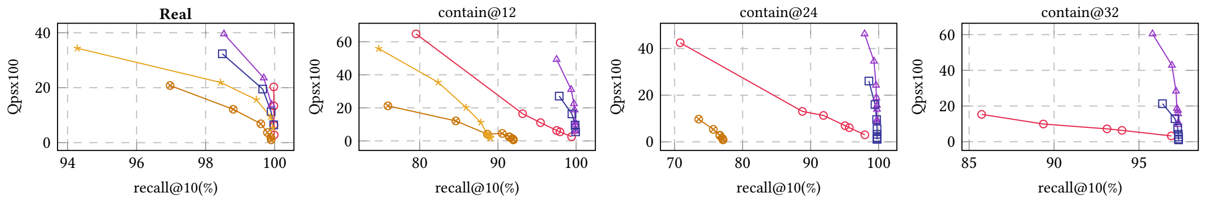
[1]github.com/KGLab-HDU/TKDE-under-review-Native-Hybrid-Queries-via-ANNS
[2]huggingface.co/datasets/Cohere/msmarco-v2.1-embed-english-v3
[3]huggingface.co/datasets/Qdrant/dbpedia-entities-openai3-text-embedding-3-large-1536-1M
[4]tripdatabase.github.io/tripclick
[5]storage.yandexcloud.net/yandex-research/ann-datasets/T2I/base.1B.fbin



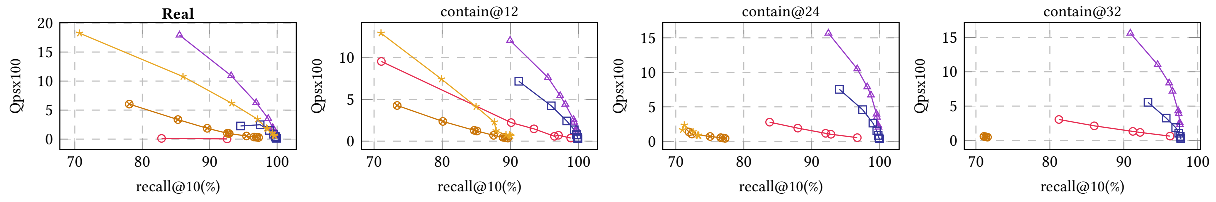
(a) SIFT



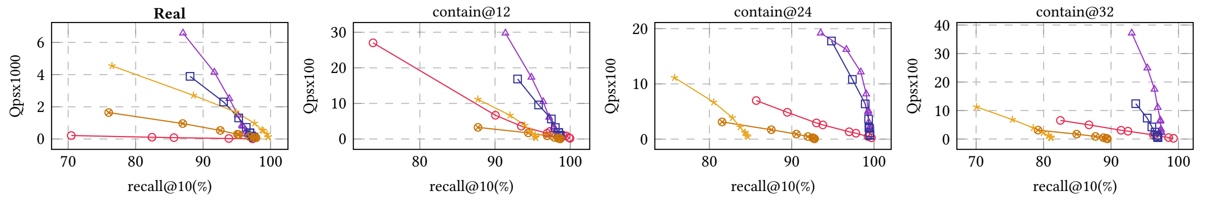
(b) MSMARCO



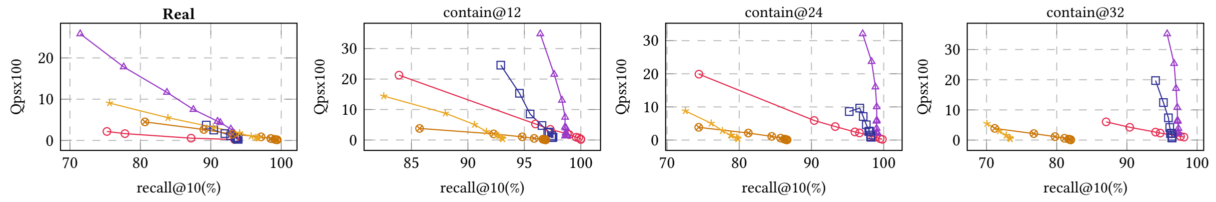
(c) PAPER



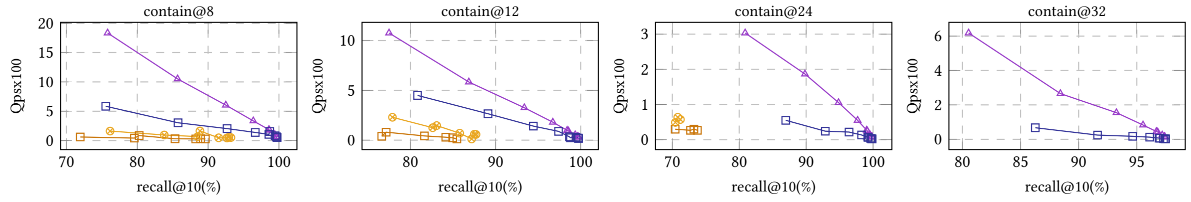
(d) OpenAI-1536



(e) LAION



(f) TripClick



(g) DEEP

Figure 9: The Tradeoff Between Accuracy and Efficiency

Table 4: Summary of Index Time (s)

	UNG	ACORN-1	ACORN-γ	ELI-0.2	ELI-2.0
SIFT	405	7	62	34	25
MSMARCO	459	29	290	148	87
PAPER	387	9	53	26	30
LAION	569	105	3238	121	63
TripClick	810	71	2656	74	45
OpenAI-1536	218	47	568	409	166
DEEP	-	912	11328	8864	3991

Table 5: Summary of Index Size (Mb)

	Base	UNG	ACORN-1	ACORN-γ	ELI-0.2	ELI-2.0
SIFT	488	186	442	485	634	382
MSMARCO	3906	233	442	485	634	382
PAPER	762	190	442	485	344	320
LAION	2309	276	2038	1820	871	406
TripClick	2990	240	1760	1570	594	339
OpenAI-1536	5853	261	442	485	782	342
DEEP	36621	-	44262	48591	85293	37902

$|L| = 12$, our algorithm achieves a 4x improvement in search efficiency over the state-of-the-art algorithm UNG on the SIFT dataset and a 10x performance improvement on the MSMARCO dataset. Moreover, our algorithm performs very stably under different $|\Sigma|$ or real label distributions. In contrast, the retrieval efficiency of the UNG algorithm decreases significantly with the increase of $|\Sigma|$, which is also verified in its paper. Our algorithm uses the elastic factor to achieve stable performance and then has a nearly 12x search speed to UNG when $|\Sigma| = 32$ on the OpenAI dataset. The search accuracy of the ACORN algorithm stuck at a bottleneck under larger $|\Sigma|$ settings, mainly due to the shortcomings of the PreFiltering strategy it adopts. Note that there is a partial performance gap between ELI-0.2 and ELI-2.0. This is because under the Zipf distribution, the elastic factor that ELI-2.0 can achieve with only double space is usually less than 0.2 when $|\Sigma|$ is large. Most importantly, as $|\Sigma|$ increases, the search accuracy of the ACORN decreases, and the search efficiency of UNG drops, which has been verified by experiments in their respective papers. However, our fixed efficiency approach ELI-0.2 has higher efficiency as $|\Sigma|$ increases. This is because a larger $|\Sigma|$ leads to a lower average selectivity of the query, while our algorithm can guarantee the search efficiency. Next, we analyze the efficiency improvement brought by more index space.

Exp-2: Index Time and Space. We compare the index time with baseline methods in Table 4. Since the space size of the partial index is affected by $|\Sigma|$ and increases monotonically with synthetic labels, we report the index time and space results of $|\Sigma| = 32$ in efficiency comparison Fig. 9. For datasets with real labels, we report the index time and space with the real label distribution. Eventually, we obtained the following results. First, we find that the indexing time of our ELI method is much more efficient than UNG and ACORN-γ. Specifically, UNG requires nearly 20x index time on the SIFT dataset and 10x on the PAPER dataset, while ACORN-γ demands on average 2x index time of our method. Although ACORN-1 is efficient in constructing indexes, the consequence is worse search efficiency due to its complete lack of label information in the indexing process. Next, we examine the index space. As illustrated in Table 5, our methods use a comparable index space to achieve the best search performance. In particular, both ACORN and ELI-2.0 use twice the HNSW space of the origin vector, while ELI-2.0 has higher search performance, which also shows that our algorithm achieves a better space-efficiency tradeoff. The UNG algorithm saves more space, but the index only accounts for a small proportion of the vector dataset size in high-dimensional cases. For instance, the index size of all the methods only occupies 1/10 of the total space on the OpenAI-1536 dataset.

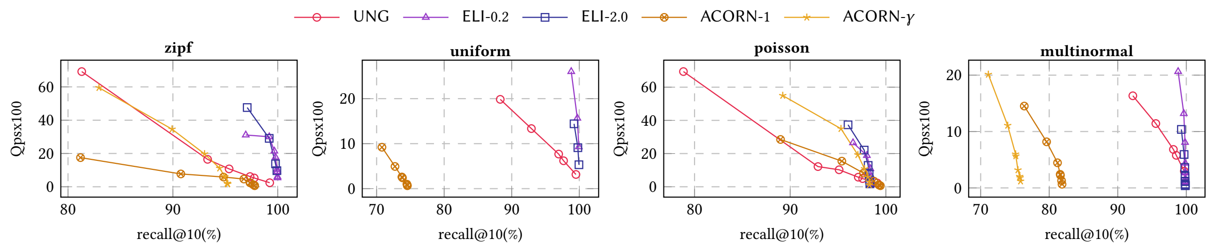
Exp-3: Test of Scalability. We also evaluate the scalability of various approaches. For this purpose, we perform experiments on the largest dataset, DEEP100M, adjusting the recall rate to compare different methods. As illustrated in Fig. 9, ACORN exhibits 4x worse performance than our proposed ELI-0.2 at a 90% recall rate and 3x worse than ELI-2.0. Additionally, ELI maintains consistent performance on large-scale datasets, while ACORN fails to reach the target 90% recall when $|\mathcal{L}|$ is large. Our approach also provides notable benefits in space efficiency and indexing time for large-scale datasets. For instance, the ELI-2.0 uses less index space and time to achieve better search efficiency than ACORN. UNG core dump during index building, failed to build the index.

Exp-4: Test of Varying Label Distribution. The label distribution affects the index selection strategy. We evaluate our methods and our baseline under various distribution settings, such as Uniform, Poisson, and Zipf data with $|\Sigma| = 12$. We use the original code of UNG to generate synchronized data. From the result in Fig. 10, we derive the following conclusion. (1) Our approach performs stably across different distribution settings. The search accuracy of ACORN varies when the distribution changes. Under Uniform and Multinormal distribution, ACORN fails to reach 80% recall while UNG and our method ELI still perform well. (2) Our methods are highly competitive in a variety of distributions. Our ELI algorithm has comparable performance to UNG at high recall for all four distributions, and has 3x-5x better search efficiency under Uniform, Poisson, Zipf, and Multinormal distributions.

Exp-5: Test of Varying Label Set Size. We test the search efficiency and index build time of a large base label set $|\mathcal{L}|$ under the Zipf distribution. We set $|\Sigma|$=64,128,256,512 to test UNG and our method ELI-0.2. Since the ACORN method fails to achieve 80% recall in these label set size settings, which is not shown in Fig. 11. From the result in Fig. 11, our method achieves nearly 800x search efficiency speed up at 99% recall when $|\Sigma| = 512$. This is because the fix efficiency feature allows ELI-0.2 to achieve higher performance with the low selectivity. The UNG suffers from the large $|\Sigma|$, the search efficiency degrades when $|\Sigma|$ grows, which also has been verified in their paper. Moreover, the index build time of ELI-0.2 is much more efficient than UNG. For example, the index time of ELI-0.2 with $|\Sigma| = 512$ is 155 second where UNG use 2091 second which is 13x lower than ELI-0.2.

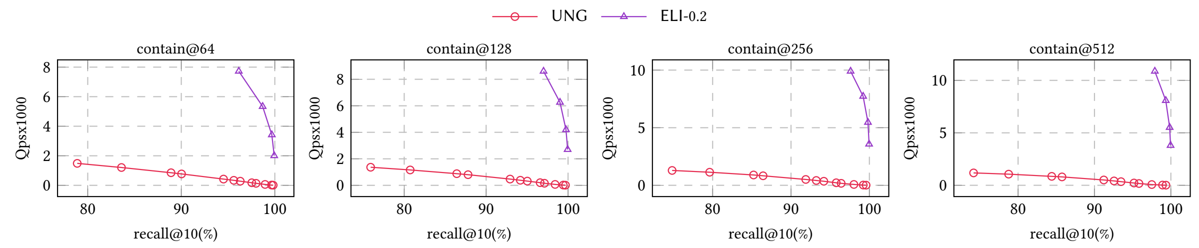
6 RELATED WORK

Vector Similarity Search. Vector similarity search has been widely studied. Most of the research focuses on approximate search [17], as exact search has a high cost in high-dimensional space [26]. Among them, c-approximate nearest neighbor search returns the result with an approximate ratio of c, which can be solved by local



(a) PAPER

Figure 10: Varying Label Distributions



(a) SIFT

Figure 11: Varying the Label Set Size $|\mathcal{L}|$

sensitive hashing (LSH) in sublinear time [7, 12, 17, 18, 25, 43, 59]. For the AKNN search problem in this paper, graph-based vector indexes [10, 11, 21, 28, 34, 37, 41, 47] are the current state-of-the-art solution, which is much more efficient than LSH according to benchmark [33]. In addition, inverted list-based indexes [2, 29] and quantization-based methods [13, 15, 16, 20, 29, 30, 55] are also widely used in different scenarios of AKNN search. For example, inverted indexes use less space, and quantization methods can speed up the distance computation of AKNN search [46, 53]. We use the most widely used HNSW algorithm [36] as the module index in this paper, and other well-optimized AKNN search libraries [28, 60] can also replace it.

Attribute-filtering AKNN search. Various ways of AKNN indexing support attribute value filtering, where the attribute can be a label, a keyword, or a numerical value. We divided them into two classes: the attribute-free and attribute-involved approach.

Attribute-Free Approach. These methods do not require attributes during index construction and determine whether a vector is filtered on the fly. Based on this, the PreFiltering and PostFiltering search strategies have been proposed. These two strategies have different performances under different queries. Database systems such as ADB [50], VBASE [58], CHASE [35], Milvus [44] etc., select different strategies based on the cost model. The ACORN is also based on the PreFiltering strategy, but builds a denser graph index to avoid the connectivity issue caused by the PreFiltering strategy. Since the above strategies completely ignore the attributes during index construction, the search accuracy and efficiency have a large gap compared to the method that involves attributes.

Attribute-Involved Approach. The attribute-involved index is constructed based on different attribute types. For range filter queries on numerical attributes, recent methods [9, 52, 54] construct a segment tree to reconstruct the query interval. SeRF [61] uses the idea of compression to compress the index corresponding to each query range interval into a compact one. For label attributes, UNG [5] uses the inclusion relationship of the label set and constructs cross-group edges to ensure that the graph index only searches the filtered vectors. Beyond the above methods, NHQ [45] takes the attribute as part of the vector similarity computation, which requires manually adjusting the weight of the two parts. The recent approach DEG [57] solves this by adjusting the graph neighbor dynamically.

The Index Selection Problem. Materialized view/index selection is a critical problem in many applications that has been studied for decades [22, 38]. Various greedy algorithms have been proposed to solve this problem, such as [22], maximizing the benefit of selecting views at each round. Different from previous studies, we study the efficiency-oriented index selection scheme, use elastic factors to model efficiency, and provide a space-based cost function, making full use of the existing vector index theory and features. Finally, we provide index selection under space constraints. In the future, we will consider index selection under changes in query workload and more advanced index selection algorithms.

7 CONCLUSION

This paper investigates the label-hybrid AKNN search problem. We begin by analyzing the limitations of existing indexing methods, which must accommodate an exponential number of label sets. To overcome this challenge, we introduce the concept of the elastic factor, which ensures that an index for a label set remains effective for its supersets, with guaranteed bounded query time. Building on this insight, we formalize the Elastic Index Selection (EIS) problem, which enables the selective construction of partial indexes while still ensuring query efficiency. We prove that the EIS problem is NP-complete and propose two optimization variants, for which we develop efficient greedy algorithms. Extensive experiments demonstrate the effectiveness and superiority of our approach. As future work, we plan to explore maintaining the result in dynamic settings

REFERENCES

[1] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2015. Cache locality is not enough: High-Performance Nearest Neighbor Search with Product Quantization Fast Scan. *Proceedings of the VLDB Endowment* 9, 4 (2015).

[2] Artem Babenko and Victor S. Lempitsky. 2015. The Inverted Multi-Index. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 6 (2015), 1247–1260.

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data.* 322–331.

[4] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning.* 97–104.

[5] 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.

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

[7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry.* 253–262.

[8] Rong-chii Duh and Martin Fürer. 1997. Approximation of k-set cover by semi-local optimization. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.* 256–264.

[9] Joshua Engels, Benjamin Landrum, Shangdi Yu, Laxman Dhulipala, and Julian Shun. 2024. Approximate Nearest Neighbor Search with Window Filters. *ICML 2024* (2024).

[10] Cong Fu, Changxu Wang, and Deng Cai. 2022. High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 8 (2022), 4139–4150.

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

[12] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data.* 541–552.

[13] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2024. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2409.09913* (2024).

[14] Jianyang Gao and Cheng Long. 2023. High-Dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations. *Proc. ACM Manag. Data* 1, 2 (2023), 137:1–137:27. https://doi.org/10.1145/3589282

[15] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[16] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 4 (2014), 744–755.

[17] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[18] Michel X Goemans and David P Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)* 42, 6 (1995), 1115–1145.

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

[20] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 12 (2013), 2916–2929. https://doi.org/10.1109/TPAMI.2012.193

[21] Yutong Gou, Jianyang Gao, Yuexuan Xu, and Cheng Long. 2025. SymphonyQG: Towards Symphonious Integration of Quantization and Graph for Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–26.

[22] Venky Harinarayan, Anand Rajaraman, and Jeffrey D Ullman. 1996. Implementing data cubes efficiently. *Acm Sigmod Record* 25, 2 (1996), 205–216.

[23] Hazar Harmouch and Felix Naumann. 2017. Cardinality estimation: An experimental survey. *Proceedings of the VLDB Endowment* 11, 4 (2017), 499–512.

[24] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 5713–5722.

[25] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[26] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing.* 604–613.

[27] Piotr Indyk and Haike Xu. 2023. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations. *Advances in Neural Information Processing Systems* 36 (2023), 66239–66256.

[28] 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).

[29] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128.

[30] Yannis Kalantidis and Yannis Avrithis. 2014. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2321–2328.

[31] Richard M Karp. 2009. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art.* Springer, 219–241.

[32] 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.

[33] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Trans. Knowl. Data Eng.* 32, 8 (2020), 1475–1488.

[34] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: Hierarchical Graph Structure Based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 15, 2 (2021), 246–258.

[35] Rui Ma, Kai Zhang, Zhenying He, Yinan Jing, X Sean Wang, and Zhenqiang Chen. 2025. CHASE: A Native Relational Database for Hybrid Queries on Structured and Unstructured Data. *arXiv preprint arXiv:2501.05006* (2025).

[36] Yu A Malkov and Dmitry A Yashunin. [n.d.]. hnswlib. https://github.com/nmslib/hnswlib.

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

[38] Imene Mami and Zohra Bellahsene. 2012. A survey of view selection methods. *Acm Sigmod Record* 41, 1 (2012), 20–29.

[39] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11 (2002), 28–46.

[40] 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.

[41] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. *Proc. ACM Manag. Data* 1, 1 (2023), 54:1–54:27. https://doi.org/10.1145/3588908

[42] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization.* Springer, 291–324.

[43] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: Solving c-Approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. *Proc. VLDB Endow.* 8, 1 (2014), 1–12.

[44] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data.* 2614–2627.

[45] 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).

[46] Mengzhao Wang, Haotian Wu, Xiangyu Ke, Yunjun Gao, Yifan Zhu, and Wenchao Zhou. 2025. Accelerating Graph Indexing for ANNS on Modern CPUs. *arXiv preprint arXiv:2502.18113* (2025).

[47] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *Proc. ACM Manag. Data* 2, 1 (2024), V2mod014:1–V2mod014:27. https://doi.org/10.1145/3639269

[48] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978.

[49] Runhui Wang and Dong Deng. 2020. DeltaPQ: lossless product quantization code compression for high dimensional similarity search. *Proceedings of the VLDB Endowment* 13, 13 (2020), 3603–3616.

[50] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: a hybrid analytical engine towards query fusion for structured and unstructured data. *Proceedings of the VLDB Endowment*

13, 12 (2020), 3152–3165.

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

[52] Yuexuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S Jensen. 2024. iRangeGraph: Improvising Range-dedicated Graphs for Range-filtering Nearest Neighbor Search. *arXiv preprint arXiv:2409.02571* (2024).

[53] Mingyu Yang, Wentao Li, Jiabao Jin, Xiaoyao Zhong, Xiangyu Wang, Zhitao Shen, Wei Jia, and Wei Wang. 2024. Effective and General Distance Computation for Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2404.16322* (2024).

[54] Mingyu Yang, Wentao Li, Zhitao Shen, Chuan Xiao, and Wei Wang. 2025. ESG: Elastic Graphs for Range-Filtering Approximate k-Nearest Neighbor Search. *arXiv preprint arXiv:2504.04018* (2025).

[55] Mingyu Yang, Wentao Li, and Wei Wang. 2024. Fast High-dimensional Approximate Nearest Neighbor Search with Efficient Index Time and Space. *arXiv preprint arXiv:2411.06158* (2024).

[56] Mingyu Yang, Wenxuan Xia, Wentao Li, Raymond Chi-Wing Wong, and Wei Wang. 2025. *Technical Report.* https://github.com/mingyu-hkustgz/LabelANN

[57] Ziqi Yin, Jianyang Gao, Pasquale Balsebre, Gao Cong, and Cheng Long. 2025. DEG: Efficient Hybrid Vector Search Using the Dynamic Edge Navigation Graph. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–28.

[58] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, et al. 2023. {VBASE}: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 377–395.

[59] Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S. Jensen. 2020. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *Proc. VLDB Endow.* 13, 5 (2020), 643–655.

[60] Xiaoyao Zhong, Haotian Li, Jiabao Jin, Mingyu Yang, Deming Chu, Xiangyu Wang, Zhitao Shen, Wei Jia, George Gu, Yi Xie, et al. 2025. VSAG: An Optimized Search Framework for Graph-based Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2503.17911* (2025).

[61] 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 (2024), 1–26.

APPENDIX

Common Used Notations

We list the commonly used notations in Table 6.

Table 6: A Summary of Notations

Notation	Description
S	A set of vectors, each with an extra label set
L_i, L_q	The base and query label set
$S(L_q)$	The dataset filtered by query label set L_q
$\|L\|$	The size of set L
$\mathcal{L}$	The collection of label sets
$\|\mathcal{L}\|$	The number of label sets in $\mathcal{L}$
Σ	The label alphabet with all possible labels
N	The cardinality of S
q	The query vector
e	The elastic factor
d	The dimensionality of S
$\delta(u, v)$	The distance/similarity between u and v
I	The AKNN search index such as HNSW
τ	The space limitation

Discussion of Various Types of Queries

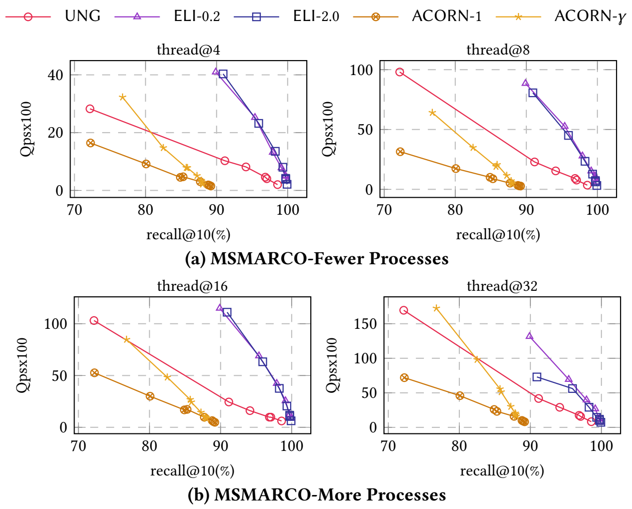
In the case of label equality—where the query requires base vectors whose label set L_i exactly matches the query label set L_q—optimal performance can be achieved by indexing each group of entries sharing the same label set. As illustrated on the left of Fig. 4, base vectors are grouped via a label-set inverted list. To support all possible label-equality queries, an index can be built for each group, and the overall dataset cardinality remains unchanged. For example, an index for $L_q = \{A\}$ is built over the base vectors x_2 and x_5. When using a graph-based index—the most efficient indexing structure to date—the overall space complexity is $O(NM)$, where M is the maximum degree of any node, constrained by edge occlusion.

For label-overlap queries with the condition $L_i \cap L_q \neq \emptyset$, optimal performance can be achieved by decomposing the query into $|L_q|$ label-containing queries, as described below. Specifically, the overlap condition is transformed into a set of label-containing queries, each with a single label. For example, a label-overlap query with $L_q = \{A, B\}$ can be answered by merging the results of the label-containing queries with $L_q = \{A\}$ and $L_q = \{B\}$. As a result, label-containing queries form the core focus of this paper.

Extra Experiments

Exp-1: Test of Varying Query Threads. The parallelism affects the search performance. We adjust the number of threads from 4 to 32 with $|\mathcal{L}| = 12$ in Fig. 12. From the result in Fig. 12, our method achieves the best search performance under various thread settings. This indicates the robustness of our method in a multi-thread search scenario. Moreover, our separate architecture requires only one sub-index to be invoked for a single query, making it more suitable in a distributed system.

Exp-2: Compare to Optimal Approach. The optimal approach build indices for all possible label-hybrid queries. We compare the search efficiency in Fig. 13 under the Zipf distribution when $|\Sigma| = 32$.

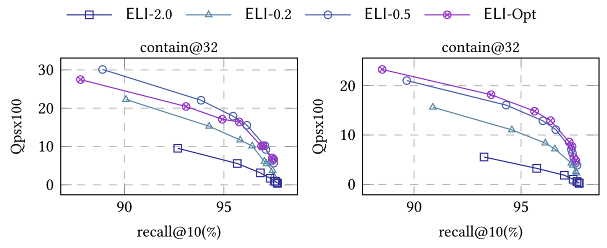


(a) MSMARCO-Fewer Processes

(b) MSMARCO-More Processes

Figure 12: Varying Number of Query Threads

From the result in Fig. 13, ELI-0.5 and ELI-0.2 achieve near-optimal search efficiency at high recall. The ELI-2.0 has 3x worse Qps than optimal, but saves more space. The index space of ELI-2.0, ELI-0.2, ELI-0.5, and ELI-opt are 383MB, 634MB, 812MB, and 1280MB, respectively. Without the top index size 192MB that must include, the ELI-2.0 only uses 1/6 of the optimal approach space to achieve acceptable search performance. Moreover, the ELI-0.5 uses a tiny more half space to achieve almost equivalent search performance, making our methods more advantageous in an efficiency-oriented scenario.



(a) MSMARC

(b) OpenAI-1536

Figure 13: Compare to the Optimal Approach