

# Elastic Index Selection for Label-Hybrid AKNN Search

Mingyu Yang<sup>1,2</sup>, Wenxuan Xia<sup>1</sup>, Wentao Li<sup>3</sup>, Raymond Chi-Wing Wong<sup>2</sup>, Wei Wang<sup>1,2</sup>

<sup>1</sup>The Hong Kong University of Science and Technology (Guangzhou),

<sup>2</sup>The Hong Kong University of Science and Technology, <sup>3</sup>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 their labels fully contain the query-label set. A naive solution builds a separate index for every query-label set, but the exponential growth of such sets makes this approach storage-prohibitive. To overcome this, we propose selectively indexing only a subset of query-label sets while still ensuring efficient processing for all queries. This is made possible by a key insight into label containment: an index built for a label set  $L$  can also serve any query whose label set  $L'$  is a superset of  $L$ , with query cost bounded by the elastic factor: the ratio between the number of vectors matching  $L$  and those matching  $L'$ . We formalize the index-selection task as a constrained optimization problem that chooses which label sets to index to satisfy space and query efficiency constraints. We prove the problem is NP-complete and propose efficient greedy algorithms for its efficiency- and space-constrained variants. Extensive experiments on real-world datasets show that our method achieves  $10\times$ – $800\times$  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:

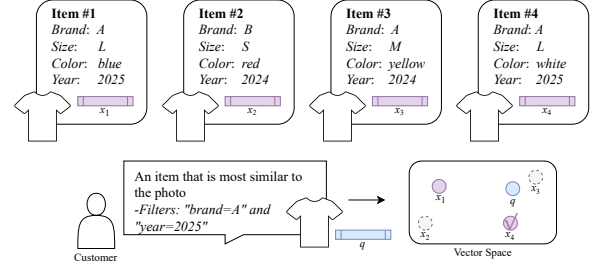
Mingyu Yang, Wenxuan Xia, Wentao Li, Raymond Chi-Wing Wong, Wei Wang. Elastic Index Selection for Label-Hybrid AKNN Search. PVLDB, 14(1): XXX-XXX, 2020.  
doi:XX.XX/XXX.XX

## 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 [48], data mining [8], face recognition [56], product search [56], and retrieval-augmented generation (RAG) for large language models [38]. 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



**Figure 1: Example of label-hybrid AKNN search in an online shopping scenario ( $k = 1$ ).** Each item is represented by an embedding vector  $x_i$  ( $i \in [1, 4]$ ) and associated with label attributes such as brand and year. A customer submits a reference photo  $q$  together with a query-label set  $L_q$ . The task is to find the item most similar to  $q$  that also satisfies the label constraint. Thus,  $x_2$  and  $x_3$  are discarded for label mismatch, and  $x_4$  is returned as the nearest neighbor.

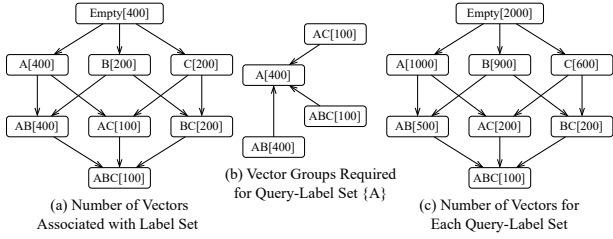
may search for items similar to a given photo while specifying label constraints like brand name and release year.

In this context, we introduce the problem of **label-hybrid  $k$ -nearest neighbor (KNN)** search, illustrated in Fig. 1. Unlike traditional KNN search, this variant considers both the vector embeddings and their associated label attributes. Formally, each entry in the database  $S$  consists of a vector embedding and a label set  $L$ . Given a query vector  $q$  and the label constraint (expressed as a query-label set  $L_q$ ), the goal is to retrieve the top- $k$  nearest neighbors of  $q$  in  $S$ , where each returned vector with label set  $L$  must contain all labels in the query-label set  $L_q$ , i.e.,  $L_q \subseteq L$ . However, due to the curse of dimensionality [32], computing exact label-hybrid KNN in high-dimensional spaces is computationally expensive. Recent work therefore studies the **label-hybrid approximate KNN (AKNN)** problem, which efficiently returns the top- $k$  approximate neighbors satisfying the label constraint, trading a small amount of accuracy for substantial performance gains [6, 46].

**Existing Solutions.** To support label-hybrid AKNN search, existing approaches often employ graph-based indexes [12, 13, 21, 23, 29, 34, 39, 40, 43, 47, 51, 54] 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 [21] 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 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.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing [info@vldb.org](mailto:info@vldb.org). Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

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



**Figure 2: Motivating example where database  $S$  and queries draw labels from the alphabet  $\{A, B, C\}$ .** Base vectors in  $S$  are grouped by their associated label sets  $L$ , each group denoted as  $\{L\}[count]$ . Arrows connect a group  $L$  to groups with its minimal supersets. For instance, Fig. 2(a) shows 400 vectors labeled only  $\{A\}$ , written as  $A[400]$ . Given an AKNN query with  $L_q = \{A\}$ , Fig. 2(b) highlights all relevant groups—those whose label sets are supersets of  $\{A\}$ —namely  $\{A, AB, AC, ABC\}$ , totaling 1,000 matching vectors that must be indexed under  $L_q = \{A\}$ . Fig. 2(c) summarizes the index sizes required for each query-label set. Supporting all possible query-label sets in the workload requires indexing 5,400 vectors—about  $2.75\times$  the original dataset size.

A base vector **matches** the query if its label set contains the query-label set and the selectivity of a query is the number of its label-matched vectors. The performance of both strategies drops sharply when selectivity is low. (1) PreFiltering suffers from reduced accuracy as removing many label-mismatched nodes sparsifies the graph, making it difficult to reach the target vectors. (2) While PostFiltering retains all nodes, it still requires computing distances between the query and many mismatched vectors, causing inefficiency. To address these challenges, existing solutions such as Milvus [50], ADB [56], VBASE [65], and CHASE [41] dynamically select between PreFiltering and PostFiltering based on the query workload and cost estimation. Yet, the limitations of these search strategies persist. Some heuristic approaches, including NHQ [51] and HQANN [57], propose fusion distances that combine both vector similarity and label matching between the query and base vectors. These methods require manual tuning of vector-label weight parameters and still perform well below the state-of-the-art.

**State-of-the-Art.** To support label-hybrid AKNN search, ACORN [46] and UNG [6] represent the state-of-the-art. ACORN extends the PreFiltering strategy to address connectivity issues under low selectivity by introducing a parameter  $\gamma$  that increases the graph density—each node has  $\gamma$  times more outgoing edges than in a standard index. This denser graph improves robustness when filtering out label-mismatched nodes during query processing. However, ACORN ignores label information during index construction, leading to potential result incompleteness.

To address this, UNG exploits label containment during graph construction. It partitions base vectors into groups by their label sets and builds a subgraph for each group. Subgraphs for a label set  $L$  are connected to those of its minimal supersets via cross-group edges, ensuring that any vector with a superset of  $L$  is reachable from the group representing  $L$ , thereby guaranteeing completeness. However, both ACORN and UNG are limited to graph-based indexes and thus lack *index flexibility*. Their performance also degrades when query selectivity is low. Experiments show significant deterioration as the size of the global label set grows, and neither method can *flexibly adjust the index structure* to meet tight storage constraints.

**Table 1: We compare our method with existing solutions along three dimensions. Search Performance measures accuracy and efficiency, validated experimentally. Index Flexibility captures the ability to support label-hybrid AKNN search without dependence on a specific index structure. Space-Performance Trade-offs assess how well the method balances indexing cost and storage efficiency.**

Feature	ELI (Our)	NHQ	UNG	ACORN	Filtered
Search Performance	★★★	★	★ <sup>†</sup>	★ <sup>†</sup>	★
Index Flexibility	✓	×	×	×	✓
Space-Performance Trade-offs	✓	×	×	×	×

**Motivation.** Given an alphabet  $\Sigma$  from which both the database  $S$  and queries draw labels, existing solutions face an exponential explosion of possible label sets. Specifically, let  $\mathcal{L}$  be the set of query-label sets in the workload (a subset of the power set of  $\Sigma$ ). A naive strategy for AKNN queries is to build a separate index for each  $L \in \mathcal{L}$ , where the index for  $L$  stores all vectors whose label sets contain  $L$ . Under this scheme, a vector with label set  $L$  must be inserted into every index whose query-label set is a subset of  $L$ . As illustrated in Fig. 2, a vector labeled  $\{ABC\}$  must appear in eight indexes:  $\{\emptyset, A, B, C, AB, AC, BC, ABC\}$ , where  $\emptyset$  denotes the top index with no label constraint ( $L_q = \emptyset$ ). This replication creates substantial overhead: empirical studies show that for  $|\Sigma| = 6-10$ , the total index entries can be  $64\times-1024\times$  larger than the original database, leading to prohibitive indexing and storage costs.

**Our Solution.** Indexing every label set in  $\mathcal{L}$  is impractical due to the sheer number of indexes and the resulting storage cost. A natural question arises: can we *index only a subset of  $\mathcal{L}$*  while still enabling efficient label-hybrid AKNN search? This is non-trivial—if a query-label set  $L \in \mathcal{L}$  is not indexed, queries with label set  $L$  may be inefficient to process. Fortunately, label containment provides a key insight: an index built for  $L$  can also serve any query whose label set  $L'$  is a superset of  $L$ . We further show that the query latency is bounded by an *elastic factor*—the ratio of vectors matching  $L'$  to those matching  $L$ . Note that a higher elastic factor implies better search efficiency. For example, in Fig. 2(c), all vectors matching  $\{AB\}$  are contained in those matching  $\{A\}$ , with an overlap ratio (or elastic factor) of at least 0.5. Thus, the index on  $\{A\}$  can efficiently answer queries for  $\{AB\}$ . The index on  $\{B\}$  has an even higher overlap with  $\{AB\}$ , yielding greater efficiency. We therefore formalize the problem of **index selection** as choosing a subset of  $\mathcal{L}$  to index while bounding both space usage and the elastic factor.

We first show that the decision version of the index-selection problem is NP-complete. We then formulate two optimization variants: (1) an **efficiency-constrained** version that minimizes index space while guaranteeing a lower bound on the elastic factor (search efficiency), and (2) a **space-constrained** version that maximizes the elastic factor under a fixed space budget. We propose a greedy algorithm for the first variant and extend it to the second, and we further describe how to update indexes in dynamic settings. Overall, our methods selectively index label sets to control space while maintaining high elastic factors for efficient label-hybrid AKNN search. Compared with existing approaches (see Table 1), our solution provides greater flexibility, higher search efficiency, and superior space-performance trade-offs.

**Contribution.** We summarize our main contributions as follows:

*Problem Analysis (§ 3).* We identify the limitations of existing label-hybrid AKNN solutions: limited flexibility in the underlying indexes and suboptimal query performance. To address these issues, we measure search performance using the *elastic factor*, which quantifies how an index built on a query-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 with label sets  $\mathcal{L}$ , can we choose a subset of  $\mathcal{L}$  so that the total index space stays within a specified bound and the elastic factor exceeds a given threshold? We prove this problem is NP-complete, showing that exact solutions are intractable.

*Index Selection Problem Solutions (§ 4).* Building on the decision version, we define two practical optimization variants of the index selection problem: (1) the efficiency-constrained variant, which minimizes space given a lower bound on the elastic factor; and (2) the space-constrained variant, 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 associated with the base vectors. Experiments show that our approach attains near-optimal search efficiency with only a 100% space overhead and outperforms competitors on large-scale datasets with large alphabets, achieving  $10\times\text{--}800\times$  speedups over state-of-the-art baselines.

Due to space constraints, some discussions and experiments are omitted, which can be found in our technical report [62].

## 2 PRELIMINARY

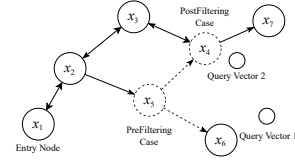
We formally define the label-hybrid approximate  $k$ -nearest neighbor search problem in § 2.1. We then review existing solutions in § 2.2.

### 2.1 Problem Statement

Each entry  $(x_i, L_i)$  in the dataset  $S$  consists of a  $d$ -dimensional base vector  $x_i \in \mathbb{R}^d$  and an associated label set  $L_i$ , which may be empty, and all possible labels are from the alphabet  $\Sigma$ . 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 label-matched 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 dataset  $S$  and a query  $(q, L_q)$ , label-hybrid KNN search returns a set  $S' \subseteq S(L_q)$  of  $k$  entries such that for every  $(x_i, L_i) \in S'$  and every  $(x_j, L_j) \in S(L_q)$ ,  $\delta(x_i, q) \leq \delta(x_j, q)$ , where  $\delta(\cdot, \cdot)$  denotes the vector distance (We use the Euclidean distance as an example).

Exact  $k$ -nearest neighbor search often suffers from the *curse of dimensionality* [32], causing traditional methods [4, 5, 45] that perform well in low-dimensional spaces to degrade greatly in high-dimensional settings. As a result, approximate  $k$ -nearest neighbor (AKNN) search has been extensively studied [1, 9, 12, 13, 16–20, 34, 36, 39, 43, 53, 55, 59, 61, 66] for its ability to improve efficiency at the cost of some accuracy. This challenge also arises in the context



**Figure 3: The Example of Filter-Based Search**

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  $\text{recall} = |\hat{S} \cap S'| / |\hat{S}|$ , where  $S'$  is the exact result set and  $\hat{S}$  is the approximate result set with  $|\hat{S}| = k$ . An effective label-hybrid AKNN solution should balance high search efficiency with high recall.

**Remark.** Label-hybrid AKNN search can be viewed as a special case of filtered nearest-neighbor search [21], where our filtering requires the label set  $L_i$  of a base vector to contain the query-label set  $L_q$ . Other variants—such as set intersection ( $L_q \cap L_i \neq \emptyset$ ) or set equality ( $L_q = L_i$ )—can be handled through simple transformations and partition-based indexing. A detailed analysis appears in §3 of our technical report [62].

### 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 [13, 29, 34, 40, 43, 47] 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.** When a graph is used as an index, search starts from a designated entry node and iteratively explores neighbors that are progressively closer to the query until the top-1 result is found. A key index design component is the edge-occlusion strategy, which guides the search toward the query while keeping the graph sparse and the node degree bounded by a constant [13, 33]. For AKNN search—where more than one result is required—beam search is typically employed. It maintains the top- $m$  closest candidates during traversal, where  $m$  is the beam width. Under ideal indexing conditions (i.e., high index quality and the query present in the dataset) [13, 33], only one extra step is needed to retrieve the next neighbor, yielding an overall search complexity logarithmic in the dataset size.

**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 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  $x_7$ .



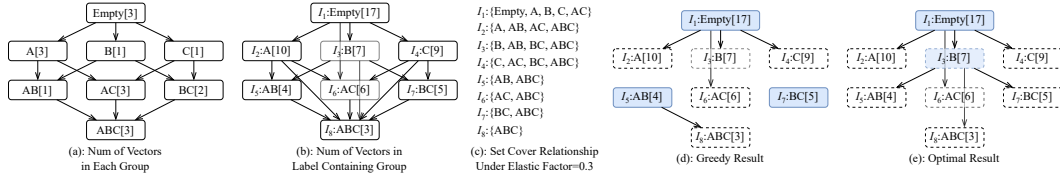


Figure 4: The Running Example of Elastic Factor and Greedy Methods

Although  $x_4$  is label-unmatched, its outgoing edge is still used for routing, enabling the algorithm to eventually reach and return  $x_7$ .

For top- $k$  queries, PostFiltering traverses incrementally, visiting successive nearest neighbors until  $k$  matching results are found. However, if most nodes are label-mismatched, the algorithm may still need to examine up to  $O(N) = O(|S|)$  nodes to obtain  $k$  results, because these filtered-out nodes must still be visited for PostFiltering. 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.

### 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. In particular, we first introduce the elastic factor, which captures how an index built on a query-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

Given an alphabet  $\Sigma$ , one could in principle build an index for every label set in the power set of  $\Sigma$ , but the number of such sets is exponential. Fortunately, index sharing can dramatically reduce the required indexes. Specifically, an index built on a query-label set  $L_2$  can serve as a substitute for another query-label set  $L_1$  if it contains all base vectors associated with  $L_1$ . This holds when  $L_2 \subseteq L_1$ , because every vector matching  $L_1$  also matches its subset  $L_2$ . For example, the group labeled  $\{A\}$  includes all entries from its superset group  $\{AB\}$ , since any vector labeled  $\{AB\}$  also contains label  $\{A\}$ . Thus, a query with label set  $\{AB\}$  can be answered using an index built on  $\{A\}$ , allowing indexes for some query-label sets to be omitted and reducing storage costs using label containment.

**Elastic Factor.** We next analyze the query efficiency provided by shared indexes. For simplicity, assume we already have a subset of indexes  $\mathbb{I} = \{I_1, \dots, I_m\}$  built on a selected subset of possible query-label sets (we discuss how to choose this subset later). Each index  $I_i \in \mathbb{I}$  contains the base vectors in  $S$  that match some query-label set  $L_i$ ; we sometimes use  $I_i$  to refer directly to these label-matched vectors. An index can answer a query with label set  $L_q$  only if it contains all base vectors matching  $L_q$ , following the label-containment rule described earlier. Intuitively, if the set  $S(L_q)$  of vectors matching the query-label set  $L_q$  is fully contained in some index  $I \in \mathbb{I}$  and overlaps heavily with  $I$ , query efficiency is high because few label-mismatched vectors remain. This intuition motivates the *elastic factor*: the maximum overlap, across all indexes in  $\mathbb{I}$  that can answer query  $q$ , between  $S(L_q)$  and the index used.

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

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

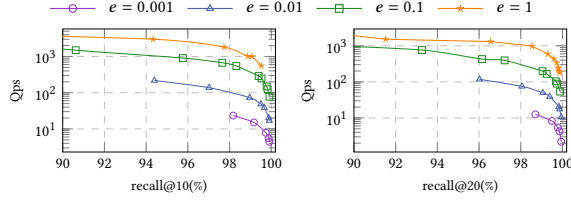
We only use indexes  $I_i$  satisfying  $S(L_q) \subseteq I_i$ , as they contain all vectors matching  $L_q$  and can thus answer query  $q$  without omission. The *elastic factor* is the maximum overlap ratio  $|S(L_q)|/|I_i|$  between  $S(L_q)$  and any such index  $I_i \in \mathbb{I}$ . When  $\mathbb{I}$  contains a single index  $I_i$ , we also denote the elastic factor between  $L_q$  and  $I_i$  as  $e(S(L_q), I_i)$ . To guarantee a non-zero factor, we include a top index  $I_\top$  built over all base vectors in  $S$  without label filtering, ensuring  $S(L_q) \subseteq I_\top$ . The elastic factor lies in  $(0, 1]$ : it is 1 when an index is built exactly on the query-label set  $L_q$  and decreases toward 0 when the chosen index contains more label-mismatched vectors.

**Example 3.2.** Figure 4(a) lists the vector groups with their associated label sets in the database  $S$ . Three vectors have no labels (Empty[3]) and three have the label set  $\{ABC\}$  [3]. Figure 4(b) shows the number of vectors matched by each possible query-label set. For example, a query with label set  $L_q = \{A\}$  matches all vectors labeled  $\{A\}$ ,  $\{AB\}$ ,  $\{AC\}$ , or  $\{ABC\}$ , totaling 10 vectors. Building index  $I_2$  on these 10 vectors (label set  $\{A\}$ ) supports not only  $L_q = \{A\}$  but also queries such as  $L_q = \{AB\}$ , since it contains all required base vectors. Figure 4(c) illustrates index sharing with an elastic factor of 0.3. Index  $I_2$  can answer the query  $L_q = \{ABC\}$  because the overlap ratio is  $3/10 = 0.3$ , whereas index  $I_1$  (label set  $\emptyset$ ) cannot, as its overlap ratio  $3/17$  is below the threshold.

**Connection to Query Efficiency.** In the extreme, one could perform filter-based search (e.g., PostFiltering) on the top index  $I_\top$ , which contains all entries and whose label set  $\{\emptyset\}$  is a subset of every query-label set. However, this is inefficient for queries with non-empty  $L_q$  because the search must traverse many label-mismatched nodes in the graph index. The inefficiency arises because the elastic factor of  $I_\top$  is typically very low for such queries. To raise the elastic factor, additional indices must be added to  $\mathbb{I}$ . The key question then becomes: when the elastic factor of  $\mathbb{I}$  for a query  $q$  is at least  $c$ , what is the resulting query cost?

To answer this question, we first analyze the expected cost of KNN search. Any index supporting top-1 nearest-neighbor search can be extended to return  $k$  results by incrementally continuing the search from each discovered neighbor until  $k$  results are found. Hence, the KNN efficiency of an index depends on the expected number of steps required to obtain  $k$  valid results. When using an index built over the entire dataset  $S$  (as in PostFiltering), the expected number of steps  $\mathbb{E}(r)$  can grow to  $O(N)$  under low query selectivity, where  $N = |S|$ , explaining its inefficiency.

In contrast, if the query is answered using an index with a constant elastic factor  $c$ , the expected number of steps to find  $k$  results



**Figure 5: Effect of the elastic factor on query efficiency (Top-10: left, Top-20: right).** We randomly generate label sets for both base and query vectors and build HNSW indexes on the SIFT100M dataset. Queries are grouped by elastic factor—0.001, 0.01, 0.1, and 1—with  $e = 1$  serving as the optimal baseline.

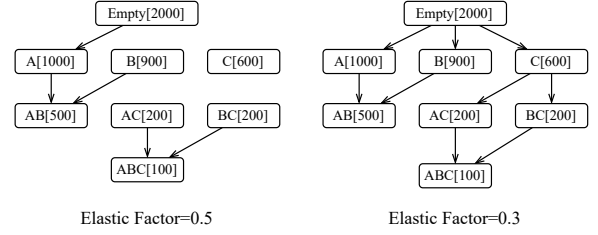
is bounded by  $O(k/c)$  [46, 60]. This bound holds because, in graph-based indexes, retrieving the next neighbor typically requires only *one* extra step under ideal conditions (e.g., slow growth or when the query exists in the dataset). Thus, when a label-hybrid AKNN search is executed on an index with elastic factor  $c$ , its asymptotic time complexity matches that of top-1 neighbor search—assuming independence between labels and vectors—up to an additional factor of  $k/c$ . We formalize this in Lemma 3.3.

**LEMMA 3.3 (EXPECTED SEARCH TIME).** *Given a label-hybrid dataset  $S$ , a query  $(q, L_q)$ , and a selected index set  $\mathbb{I}$ , let  $O(C)$  denote the expected time to retrieve the top-1 neighbor from a graph index. If the elastic factor satisfies  $e(S(L_q), \mathbb{I}) \geq c$  for some constant  $c \in (0, 1]$ , then the expected time to obtain the top- $k$  label-hybrid AKNN results with PostFiltering is  $O\left(C + \frac{k}{c}\right)$ .*

*Proof sketch.* Assume that (i) after the entry stage (whose cost is included in  $C$ ) to find the top-1 nearest neighbor, each successive neighbor can be retrieved with amortized constant work, as is typical for graph-based AKNN indexes, and (ii) conditioned on distance ordering, at least a fraction  $c$  of the visited candidates match the query labels (ensured by  $e(\cdot, \cdot) \geq c$ ). Because PostFiltering outputs a vector only if it matches the query-label set, the number of examined items  $T$  until  $k$  matches follows a negative-hypergeometric (or geometric) distribution with success probability at least  $c$ , giving  $\mathbb{E}[T] \leq k/c$ . Adding the entry cost  $C$  to these incremental steps yields the bound  $O(C + k/c)$ .

**Remark on Lemma 3.3.** The above result provides an *expected* bound under two standard assumptions: (1) locating the top-1 result costs  $O(C)$ , and enumerating additional neighbors requires amortized constant work; (2) Label-matched and mismatched vectors are assumed to be uniformly distributed in the graph and independent of the labels, ensuring that at least a  $c$ -fraction of visited items satisfy the label constraint (as guaranteed by the elastic factor). This bound is not a worst-case guarantee, as these assumptions may not always hold for graph indexes such as HNSW. Nevertheless, the elastic factor remains a useful control knob for expected performance, because the additional time cost scales directly with  $k/c$ . Our experiments (Fig. 5) confirm this monotonic relationship: larger elastic factors yield higher query throughput (QPS) at fixed recall, and the observed performance drop as  $c$  decreases matches the theoretical prediction.

**Empirical Studies.** To further validate Lemma 3.3, we measure the practical efficiency of PostFiltering search under varying elastic factors on the 100M subset of the SIFT1B dataset<sup>1</sup>, denoted



**Figure 6: Example of the EIS Problem**

SIFT100M. In the classical PostFiltering strategy, the elastic factor is fixed because it uses only the top index containing all base vectors. Here, we add additional indexes to  $\mathbb{I}$  to vary the elastic factor and report results in Fig. 5. As shown, higher elastic factors yield better efficiency (measured in QPS), consistent with the lower time complexity predicted by theory. When the factor reaches 1, search achieves optimal efficiency—equivalent to indexing only the query-matched data. Efficiency decreases sublinearly as the elastic factor drops: for example, reducing the factor to  $\frac{1}{10}$  lowers throughput by only  $2\times$  at 98% recall with  $k=10$ . This occurs because the added cost scales with  $k$ , while the top-1 search still runs in  $O(C)$ , usually  $O(\log N)$ . The gap widens for larger  $k$  (e.g., a  $3\times$  slowdown at  $k=20$  and 98% recall), but in most applications  $k$  is small relative to  $N$ . Thus, as long as the elastic factor remains constant, the overall time cost stays bounded.

### 3.2 Problem Definition

We have analyzed the impact of the elastic factor on search efficiency for a *given* index set  $\mathbb{I}$ . When using  $\mathbb{I}$ , the expected query cost is directly tied to the elastic factor of  $\mathbb{I}$  for each query. If the elastic factor for every query in the workload is guaranteed to exceed a constant lower bound  $c$ , then the AKNN search cost is bounded by an additional  $O(k/c)$ . This establishes a clear relationship between index cost (the total size of indexes in  $\mathbb{I}$ ) and query efficiency (captured by the elastic factor). Building on this, among *all subsets of indexes* that can correctly answer the query workload, we seek a subset  $\mathbb{I}$  that is both small and yields a large elastic factor, ensuring strong query performance. This motivates the formal definition of the **Elastic Index Selection (EIS)** problem.

**Definition 3.4 (Fixed Efficiency Index Selection (EIS)).**

**Input** A label-hybrid dataset  $S$ , a query workload with label sets  $\mathcal{L} = \{L_1, \dots, L_n\}$ , and a universal index set  $\mathcal{I} = \{I_1, \dots, I_n\}$ , where each index  $I_i = S(L_i)$  is built over the base vectors matching the query-label set  $L_i$  with cost  $|I_i|$ . Let  $\tau$  and  $c$  be non-negative real numbers.

**Output** Decide whether there exists a subset  $\mathbb{I} \subseteq \mathcal{I}$  such that  $e(S(L_i), \mathbb{I}) \geq c$  for every  $L_i \in \mathcal{L}$  and the total cost satisfies  $\sum_{I \in \mathbb{I}} |I| \leq \tau$ .

The EIS problem is a *decision* problem: determine whether a subset of indexes from the universal set  $\mathcal{I}$  can be chosen such that (1) the total space usage is within a user-specified bound  $\tau$ , and (2) the elastic factor—which governs query efficiency—is at least a given threshold  $c$ . Figure 6 illustrates this setting. In Section 4, we present the *optimization* version, where the goal is either to minimize space cost while satisfying a required elastic-factor threshold or to maximize the elastic factor subject to a space budget.

<sup>1</sup><http://corpus-texmex.irisa.fr>

**EXAMPLE 1.** The EIS problem under different elastic-factor thresholds,  $c = 0.5$  and  $c = 0.3$ . When  $c = 0.5$ , the top index  $I_T$  (built on the empty label set) can serve queries with  $L_q = \{A\}$  and  $L_q = \emptyset$ , since their overlaps exceed 0.5. It cannot, however, support  $L_q = \{B\}$ , whose overlap is only 0.45. With a more relaxed threshold of  $c = 0.3$ ,  $I_T$  also supports  $L_q = \{B\}$  and  $L_q = \{C\}$ , but it still cannot answer  $L_q = \{AB\}$ , whose overlap is 0.25, unless  $c < 0.25$ . Finally, when  $c = 0$ , the top index can serve all queries regardless of label constraints.

**Remark on Query Workload.** In the problem definition, we assume a query workload with query-label sets  $\mathcal{L}_q = \{L_1, \dots, L_n\}$ . This assumption is justified for two reasons: (1) It generalizes the traditional setting where  $\mathcal{L}_q$  includes all subsets of the alphabet  $\Sigma$  (i.e., the power set). (2) It enables a workload-driven approach:  $\mathcal{L}_q$  can reflect historical query patterns, allowing index selection and optimization to match realistic query behavior.

Although the total number of possible query-label sets can reach  $2^{|\Sigma|}$  in the worst case, the effective number is typically far smaller due to mutual exclusivity among certain labels. For example, a product has only one brand name, making brand labels mutually exclusive. Our formulation explicitly captures such cases, ensuring that incompatible label combinations—like different brand names—do not co-occur in  $\mathcal{L}_q$ . We also assume the workload always includes the empty set  $\emptyset$ , representing queries with no label constraints and thus matching all base vectors. For simplicity, we treat the cost of this top index as zero, i.e.,  $|I_{\text{top}}| = 0$ .

**Remark on Index Size/Cost.** The cost of each index  $I_i$  (built for query-label set  $L_i$ ) reflects the space it occupies. For graph-based indexes, this cost can be approximated by the number of vectors it contains, since node degree is typically bounded by a constant [13, 33]. In practice, each node maintains  $M$  edges to support efficient memory access, where  $M$  is a user-defined parameter [42]. Thus, the overall space usage of an index can be estimated as the product of its vector count and  $M$ . In our study, however, we measure index size/cost  $|I|$  simply by the number of vectors it contains.

**Index Selection At Query Time.** At query time, a label set  $L_q$  may be served by multiple indexes in the selected set  $\mathbb{I}$ . For example, a query with  $L_q = \{AB\}$  can be answered by the index for  $\{A\}$  or for  $\{B\}$  if both are present in  $\mathbb{I}$ . In such cases, our method selects the index that yields the maximum elastic factor with  $S(L_q)$ . This choice improves query efficiency because greater overlap means fewer label-mismatched vectors in the index, reducing the cost of scanning irrelevant entries.

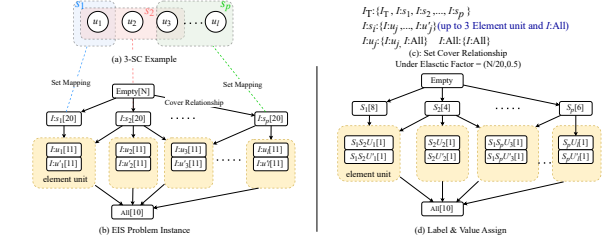
### 3.3 Problem Hardness

We proved that the EIS problem is NP-complete in Theorem 3.5, underscoring the computational challenges inherent in solving it.

**THEOREM 3.5.** *The EIS problem is NP-complete.*

**PROOF.** Given a dataset  $S$ , a query workload with label sets  $\mathcal{L} = \{L_1, \dots, L_n\}$ , and a universal index set  $\mathcal{I} = \{I_1, \dots, I_n\}$  (where each  $I_i \in \mathcal{I}$  is built on a label set  $L_i \in \mathcal{L}$ ), the EIS problem asks whether there exists a subset  $\mathbb{I} \subseteq \mathcal{I}$  such that for every query-label set  $L_q \in \mathcal{L}$  we can find an index  $I \in \mathbb{I}$  with elastic factor  $e(S(L_q), I)$  greater than a threshold  $c$ , while the total cost satisfies  $\sum_{I \in \mathbb{I}} |I| \leq \tau$ .

The problem is in NP as we can verify in polynomial time that (1) the total cost  $\sum_{I \in \mathbb{I}} |I|$  does not exceed the budget  $\tau$ , and (2) each



**Figure 7: Proof of NP-completeness.** Set  $D = b = 10$ , so  $A = b + 1 = 11$ ,  $B = 2b = 20$ , and  $N = 50 > 4b$ . Then  $c = \frac{B}{N} = \frac{20}{50} = 0.4$ . Thus we have 10 vectors in  $I_{\text{all}}$  and 50 vectors in  $I_T$ . Each element index  $I_{u_i}$  is duplicated with a dummy index  $I_{u'_i}$ , and each subset index  $I_{s_j}$  connects to at most three element indexes  $I_{u_i}$  or  $I_{u'_i}$ . Moreover,  $|I_{s_j}|/|I_T| = 0.4 = c$ ,  $|I_{u_i}|/|I_T| = \frac{11}{50} = 0.22 < c$ ,  $|I_{u_i}|/|I_{s_j}| = \frac{11}{20} > c$ .

query-label set is covered by some selected index with elastic factor at least  $c$ . Therefore, EIS lies in NP. To show NP-completeness, we reduce from the NP-complete **3-Set Cover (3-SC)** problem [10, 37].

**Definition 3.6 (3-Set Cover (3-SC)).**

**Input** A universal set  $\mathcal{U} = \{u_1, u_2, \dots, u_p\}$  of  $p$  elements and a collection  $\mathcal{S} = \{s_1, s_2, \dots, s_l\}$  of  $l$  subsets, where each  $s_i$  has size at most 3 and  $\bigcup_{i=1}^l s_i = \mathcal{U}$ . Let the selection budget be  $k$ .

**Output** Determine if there exists a subset  $\mathbb{S} \subseteq \mathcal{S}$  of size at most  $k$  such that  $\bigcup_{s_i \in \mathbb{S}} s_i = \mathcal{U}$ .

**Construction.** Given a 3-SC instance  $(\mathcal{U}, \mathcal{S}, k)$ , we construct a corresponding EIS instance (see Fig. 7). For each element  $u_i \in \mathcal{U}$ , we create an **element index**  $I_{u_i}$  of size  $|I_{u_i}| = A$ . For each subset  $s_j \in \mathcal{S}$ , we create a **subset index**  $I_{s_j}$  of size  $|I_{s_j}| = B$ . We also add two special indexes: a **top index**  $I_T$  (Empty[ $N$ ] in Fig. 7) containing all base vectors with size  $|I_T| = N$ , and a **bottom index**  $I_{\text{all}}$  (All[ $D$ ] in Fig. 7) containing entries matching all labels with size  $|I_{\text{all}}| = D$ .

Each index corresponds to the vectors matching the query-label set on which it is created. As noted earlier, the top index  $I_T$  is required to support AKNN search and always exists; For simplicity, we ignore its cost. The costs  $A, B, D$  and the elastic-factor threshold  $c$  are then controlled by defining the query-label sets  $\mathcal{L} = \{u_1, \dots, u_p\} = \mathcal{U}$  and arranging the distribution of vectors associated with these labels to satisfy the following conditions.

- (1) Each subset index  $I_{s_j}$  covers an element index  $I_{u_i}$  with overlap ratio at least  $c$  (i.e.,  $I_{u_i} \subseteq I_{s_j}$ ,  $|I_{u_i}|/|I_{s_j}| \geq c$ ) if and only if  $u_i \in s_j$ .
- (2) The top index  $I_T$  covers every subset index  $I_{s_j}$  with overlap ratio  $c$ , while its overlap with any element index  $I_{u_i}$  is less than  $c$ .
- (3) To cover element/label set  $u_i$ , selecting the element index  $I_{u_i}$  incurs higher cost than selecting any subset index  $I_{s_j}$ , where  $u_i \in s_j$ .

**Feasibility.** We next show how to make the above conditions feasible. Set the cost of  $I_{\text{all}}$  to  $D = b > 3$  (as in Fig. 7(b), where  $b = 10$  and  $I_{\text{all}}$  contains vectors covering all labels in  $\mathcal{L}$ ). Let  $A = b + 1$ ,  $B = 2b$ , and set the elastic-factor threshold to  $c = \frac{2b}{N}$  with  $N > 4b$ : If  $N \leq 4b$ , we pad  $I_T$  with vectors that carry no labels to increase  $N$  (as in Fig. 7(b), where  $N > 40$ ). Each subset index  $I_{s_j}$  is created on the query-label set  $s_j \in \mathcal{U}$ .

Each element index  $I_{u_i}$  is created from the  $b$  vectors in  $I_{\text{all}}$  plus one additional vector whose label set is  $s_1 \dots s_x u_i$ , where  $s_1, \dots, s_x$  are the subsets containing  $u_i$ , giving  $|I_{u_i}| = b + 1 = A$ . For this

additional vector with label set  $s_1 \cdots s_x u_i$ , we create another vector with label set  $s_1 \cdots s_x u'_i$ , where  $u'_i$  is a duplicate label equivalent in meaning to  $u_i$  (i.e., if  $u_i \in s_j$  then  $u'_i \in s_j$ ). Accordingly, we introduce another element index  $I_{u'_i}$  of size  $b + 1$  ( $b$  vectors from  $I_{\text{all}}$  plus one vector with labels  $s_1 \cdots s_x u'_i$ ).

Since each subset  $s_j$  contains at most three elements  $u_i$  (and their duplicates  $u'_i$ ), it covers at most  $3 \times 2$  vectors of the form  $s_1 \cdots s_x u_i$  or  $s_1 \cdots s_x u'_i$ , together with the  $b$  vectors in  $I_{\text{all}}$ . We then append up to  $2b - 6$  additional vectors labeled  $s_j$  so that the total number of vectors matching  $s_j$  equals  $B = 2b$ .

**Conditional Checking.** Note that  $c = \frac{2b}{N} < \frac{2b}{4b} = \frac{1}{2}$ . For Condition (1), when  $u_i \in s_j$ , the overlap between  $I_{u_i}$  and  $I_{s_j}$  equals  $|I_{u_i}|$ , so the overlap ratio is  $\frac{|I_{u_i}|}{|I_{s_j}|} = \frac{A}{B} = \frac{b+1}{2b} > \frac{1}{2} > c$ . For Condition (2),

for each  $s_j$ , the overlap ratio with  $I_T$  is  $\frac{|I_{s_j}|}{|I_T|} = \frac{B}{N} = \frac{2b}{N} = c$ . For each  $u_i$ , it is  $\frac{|I_{u_i}|}{|I_T|} = \frac{A}{N} = \frac{b+1}{N}$ . Since  $2b > b + 1$  and  $b > 3$ , we have  $\frac{b+1}{N} < c$ , satisfying the condition. For Condition (3), to cover vectors containing label  $u_i$ , selecting a subset index  $I_{s_j}$  with  $u_i \in s_j$  costs  $B = 2b$ . Selecting the element index  $I_{u_i}$  requires also selecting  $I_{u'_i}$ , for a total cost of  $2A = 2(b + 1)$ . As  $2b < 2(b + 1)$ , choosing  $I_{s_j}$  is cheaper, so this condition holds.

**Correctness.** Note that  $I_T$  must always be selected but incurs zero cost, while  $I_{\text{all}}$  need not be selected since it is already covered by other indexes. We set the space budget  $\tau = k \cdot B$ , which permits selecting at most  $k$  subset indexes.

**3-SC  $\Rightarrow$  EIS** If the 3-SC instance admits a cover  $\mathbb{S} \subseteq \mathcal{S}$  with  $|\mathbb{S}| \leq k$ , select the corresponding subset indexes  $\{I_{s_j} : s_j \in \mathbb{S}\}$ . Each  $I_{s_j}$  is created on the query-label set  $s_j$  and covers all vectors whose query-label set is  $u_i$  for every  $u_i \in s_j$ . The overlap ratio is at least  $c$ , so the elastic factor between  $I_{s_j}$  and both  $s_j$  and  $u_i$  is at least  $c$ . This satisfies the EIS constraints because  $\mathcal{S} = \mathcal{U}$ .

**3-SC  $\Leftarrow$  EIS** Conversely, suppose the EIS instance admits a feasible selection  $\mathbb{I}$ . By Condition (3), any  $I_{u_i} \in \mathbb{I}$  can be replaced by some  $I_{s_j}$  containing  $u_i$  without violating constraints or increasing total cost. Thus we may assume the solution consists solely of subset indexes after such replacements. Let  $\mathbb{S} = \{s_j : I_{s_j} \in \mathbb{I}\}$ . Because every  $I_{u_i}$  must be covered by some subset index (with overlap ratio at least  $c$ ), we have  $\bigcup_{s_j \in \mathbb{S}} s_j = \mathcal{U}$ . The budget enforces  $|\mathbb{S}| \leq \tau/B = k$ , yielding a valid 3-SC solution.  $\square$

The EIS problem is a decision problem with a yes/no answer. In practice, however, one often seeks to optimize either space usage or query time, as these are the key factors in deploying AKNN search. We therefore study two optimization variants of EIS: (1) **Efficiency-constrained variant** that minimizes total space while enforcing a lower bound on the elastic factor and (2) **Space-constrained variant** that maximizes the elastic factor subject to a given space budget. Because EIS is NP-complete, both variants are NP-hard, making exact solutions computationally infeasible in practice. We address these variants using greedy heuristics.

## 4 PROBLEM SOLUTION

We first present a greedy algorithm for the efficiency-constrained EIS problem, then extend it to the space-constrained variant and conclude with discussions.

---

### Algorithm 1: Greedy Algorithm

---

**Input:** The Universal Index Set  $\mathcal{I}$ , The Elastic Factor  $c$

**Output:** The Selected Index Set  $\mathbb{I}$

```

1  $\mathbb{I} \leftarrow \{I_T\};$ 
2 while Stop Condition Not Met do
3    $I \leftarrow I \in \mathcal{I} \setminus \mathbb{I}$  such that  $B(I, \mathbb{I})$  is maximum;
4    $\mathbb{I} \leftarrow \mathbb{I} \cup \{I\};$ 
5   if For any  $L_q \in \mathcal{L}$  that  $e(S(L_q), \mathbb{I}) \geq c$  then
6     Break;
7 return  $\mathbb{I};$  // Selected Index Set
```

---

### 4.1 Method for Efficiency-constrained Variant

This optimization version of the EIS problem resembles Definition 3.4, but aims to compute the following output:

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

Since this variant of the EIS problem is NP-hard, we employ a greedy algorithm to obtain an approximate solution. The basic idea is as follows. Given a collection of indexes  $\mathcal{I}$  (corresponding to a query workload with label sets  $\mathcal{L}$ ), each index  $I \in \mathcal{I}$  has an associated space cost, measured by the number of matched vectors it contains. We define a benefit function for each index and iteratively select the index with the highest benefit. Because adding more indexes can reduce the overall elastic factor, selection must be performed carefully to satisfy the constraint until all query-label sets in  $\mathcal{L}$  are covered. A key preprocessing is that we always include the top index  $I_T$  in the solution. As noted in Section 3.2, this guarantees that every query has at least one index covering all of its matched vectors, ensuring correctness. Thus the answer set  $\mathbb{I}$  is initialized with  $I_T$ . We then define the benefit of adding a candidate index  $I \in \mathcal{I} \setminus \mathbb{I}$  to the current solution  $\mathbb{I}$  as  $B(I, \mathbb{I})$ , given by:

**Definition 4.1.** Let  $\mathbb{C} \subset \mathcal{I}$  be the set of indexes already covered by the selected set  $\mathbb{I}$  with overlap ratio at least  $c$ , where  $c$  is the elastic-factor threshold:  $\mathbb{C} = \{I_i \in \mathcal{I} \mid \exists I_j \in \mathbb{I} : (I_i \subseteq I_j) \wedge (|I_i|/|I_j| \geq c)\}$ . Let  $\mathbb{C}'$  be the set of indexes newly covered by adding  $I$ , also with overlap ratio at least  $c$ :  $\mathbb{C}' = \{I_i \in \mathcal{I} \setminus \mathbb{C} : (I_i \subseteq I) \wedge (|I_i|/|I| \geq c)\}$ . The benefit of adding  $I$  to  $\mathbb{I}$  is then

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

The benefit captures two aspects: (i) the number of previously uncovered indexes that the candidate index  $I$  can cover (with overlap ratio at least  $c$ , i.e., the set  $\mathbb{C}'$ ), and (ii) the actual overlap ratio between  $I$  and each newly covered index  $I_i \in \mathbb{C}'$ . In other words, the benefit is the total overlap ratio of  $I$  with all indexes in  $\mathbb{C}'$ .

**Algorithm.** We implement the greedy method in Algorithm 1. The procedure is as follows. First, we add the top index  $I_T$  to the answer set  $\mathbb{I}$  (Line 1). At each iteration, we select the next candidate index—each candidate  $I$  corresponds to a query-label set  $L_q$  in the workload—based on its benefit score (Lines 3–4). The process stops when all label sets in  $\mathcal{L}$  are covered, yielding a valid solution (Lines 5–6), or when the elastic-factor constraint is violated (Line 5).

**Example 4.2.** Using the dataset in Fig. 4, we evaluate the benefit of different index choices, with Table 2 reporting the benefit ratio



**Table 2: The Benefits of Each Candidate Index in Each Round**

	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

for each candidate index. Note that the top index  $I_1 = I_T$  is always selected first regardless of its unit benefit. Selecting  $I_1$  covers the vectors in  $I_2, I_3, I_4, I_6$ , and itself (with overlap ratio at least the elastic-factor threshold  $c = 0.3$ ), for a total of  $17 + 10 + 7 + 9 + 6 = 49$  vectors at a cost of 17, yielding a unit benefit of  $49/17 \approx 2.88$ . After  $I_1$  is selected, the benefits of the remaining candidates change. Initially,  $I_2$  covers  $I_2, I_5, I_6$ , and  $I_8$ , i.e., 23 vectors at a cost of 10, for a unit benefit of 2.3. In the second round, as  $I_1$  already covers  $I_2$  and  $I_6$ ,  $I_2$  can now cover only  $I_5$  and  $I_8$ , reducing its unit benefit to 0.7. The greedy algorithm next selects the index with the highest updated benefit (Fig. 4(d)), which is  $I_5$ . At this point,  $I_1$  and  $I_5$  cover all vectors except those in  $I_7$ . The third round selects  $I_7$ , adding a cost of 5. The greedy solution therefore has a total cost of  $17 + 4 + 5 = 26$ , which is not optimal. As shown in Fig. 4(e), the optimal solution instead selects  $I_3$  in the second round. Although  $I_3$  covers only  $I_5, I_7$ , and  $I_8$  with a lower unit benefit of 1.71 (due to overlap with  $I_1$ ), it achieves full coverage using just two indexes,  $I_1$  and  $I_3$ , for a total cost of  $17 + 7 = 24$ , outperforming the greedy approach.

**Implementation Details and Cost Analysis.** To make the greedy method practical, we first create an index in  $\mathcal{I}$  for each query-label set in  $\mathcal{L}$ . In theory, there can be up to  $2^{|\Sigma|}$  query-label sets—and thus the same number of indexes—in the worst case. However, given a query workload  $\mathcal{L}$ , we only need to compute the sizes of  $O(\min(2^{|\Sigma|}, |\mathcal{I}|))$  indexes during preprocessing, where  $|\mathcal{I}| = |\mathcal{L}|$ . When the labels in the alphabet  $\Sigma$  follow a power-law distribution (as in many real-world datasets), this preprocessing cost is empirically a small fraction of the total runtime. For other label distributions, however, the overhead can hinder scalability for large or dense label sets. To handle such cases, following [27], we estimate the size of certain indexes in  $\mathcal{I}$  under a query-label set  $L_q$  by sampling or advanced cardinality-estimation techniques [28], without fully materializing them. Once index sizes are approximated, we can efficiently compute the benefit of each candidate and apply the greedy algorithm. To improve performance, we maintain a max-heap to track the index with the highest benefit at each iteration. Because selecting an index can affect the benefits of up to  $|\mathcal{I}|$  related indexes, we check whether the heap’s top entry has become stale; if so, we update its benefit and reinsert it. This strategy keeps the practical runtime well below  $O(|\mathcal{I}|^2)$  by incrementally checking and updating index benefits.

## 4.2 Method for Space-Constrained Variant

In the previous section, we addressed the efficiency-constrained index-selection problem. We now turn to selecting indexes that maximize query efficiency while satisfying a space constraint. This optimization variant of the EIS problem follows Definition 3.4, but focuses on computing the following output:

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

**Insight.** We observe that the elastic-factor threshold  $c$  is monotone for the EIS problem. Specifically, if an index set  $\mathbb{I}$  satisfies a threshold of 0.5 for a query workload  $\mathcal{L}$ , it also satisfies any smaller threshold  $c' < 0.5$ . This holds because  $c$  bounds the overlap between the indexes in  $\mathbb{I}$  and any label set in  $\mathcal{L}$ , requiring the overlap to be at least  $c$ . Thus, if the stricter bound  $c$  is met, any looser bound  $c'$  is automatically satisfied.

**Method.** Leveraging this property, we reduce the space-constrained problem to its efficiency-constrained counterpart and apply a binary search to find a set  $\mathbb{I}$  whose elastic factor exceeds a given  $c$  while keeping total cost within the budget  $\tau$ . We perform the binary search over  $c \in [0, 1]$  in decreasing order. At each step, for a candidate value of  $c$ , we invoke Algorithm 1 to check if the returned index set  $\mathbb{I}$  has total cost at most  $\tau$ . The search stops once such a set is found, yielding the maximum achievable elastic factor under the space constraint. Overall, this procedure requires  $O(\log \frac{1}{\epsilon})$  calls to Algorithm 1, where  $\epsilon$  is the resolution for the binary search (set to 0.001 in our paper). In practice, the overhead of combining binary search with the greedy method is negligible—typically under 1% of the index-construction time, or about 1–2 seconds even for large workloads  $\mathcal{L}$ .

## 4.3 Discussions

**Limitation.** For either efficiency- or space-constrained EIS problem, our greedy methods output a subset of indexes  $\mathbb{I}$  from the universal index set  $\mathcal{I}$ , where each index corresponds to a query-label set in the workload  $\mathcal{L}$ . The efficiency of our approach for AKNN search stems from controlling either space or query time through index sharing based on label-containment relationships. In extreme cases, however, index sharing may be ineffective. For example, if the query workload consists of label sets that are completely disjoint, there is no containment relationship between them. We must then either create a separate index for each disjoint query-label set (resulting in high space cost) or build a single index covering all vectors in the database (resulting in a small elastic factor). Fortunately, in real-world scenarios, when query-label sets are disjoint, their base vectors are typically disjoint as well, so building an index for each query-label set remains affordable, requiring only  $O(N)$  space. By contrast, algorithms such as UNG, ACORN, and FILTER-DISKANN are unaffected by this limitation as they do not rely on index sharing and may perform better in such cases.

**Extensions to Dynamic Scenarios.** In real-world deployments, both the database vectors and the query workload may evolve over time. We describe how our method accommodates such updates.

**Database vector dynamics.** Vectors can be inserted into or deleted from the database. We focus on insertions, since deletions can be handled by marking a vector as ignored and continuing the search until all results are found [12, 34, 42]. When new vectors arrive, we insert each into every index  $I_i$  (built on query-label set  $L_i$ ) whose labels match the vector. For queries, we continue using the original selected index set  $\mathbb{I}$ , but periodically check for data changes and update the indexes once 10–20% new vectors have been added.



*Query workload dynamics.* When the query workload introduces new label sets, we first collect all vectors whose label sets match the new queries. We then select an index from  $\mathbb{I}$  that serves the new query-label set. This is achieved by choosing the index built on a label set  $L$  (where  $L$  is a subset of the query-label set) that maximizes its overlap with the collected vectors. At query time, we decide whether to scan the collected vectors directly or to use the existing index based on the cost model. We also periodically adjust the selected indexes to maintain efficiency.

*Index updates.* For both types of dynamics, we postpone structural updates to  $\mathbb{I}$  and process them in batches. (1) For the efficiency-constrained variant: We maintain the current set  $\mathbb{I}$  as long as it satisfies the elastic-factor constraint. If vector insertions or newly created indexes (from new queries) violate this constraint, we reapply Algorithm 1 to add the necessary indexes to enlarge  $\mathbb{I}$ . (2) For the space-constrained variant: We use the existing  $\mathbb{I}$  and its current elastic factor as the starting point. If additional space is available, we attempt to increase the target elastic factor. Otherwise, we lower the target, prune redundant indexes to free space, and continue the binary-search process.

Throughout these updates, we support non-blocking operations: The index set need not be rebuilt from scratch, and incoming queries are routed dynamically to the index with the highest elastic factor, ensuring uninterrupted service. Empirical results in Section 5 confirm the efficiency of this update strategy.

## 5 EXPERIMENT

### 5.1 Experiment Settings

**Datasets.** We evaluate on eight publicly available vector datasets (see Table 3) that are standard benchmarks for AKNN search, such as SIFT and DEEP, as well as datasets generated by state-of-the-art embedding models such as MSMARCO, OpenAI-1536 [61], TripClick, and Arxiv [31]. To further examine scalability, we sample 100M vectors from the Deep1B [15, 61]. For label annotations, datasets PAPER, LAION, TripClick, OpenAI-1536, and Arxiv supply real labels or metadata. For others, we synthesize labels with diverse distributions following prior work [6, 21]. Label sources are grouped into three categories:

- (1) **Real labels (PAPER, TRIPCLICK, ARXIV):** For PAPER, each vector is labeled by the paper’s venue, research field, and institution type. For TRIPCLICK, we use the 28 most frequent clinic categories, consistent with prior works [6, 46]. For ARXIV, we assign the main subject categories of each vector as its label.
- (2) **Metadata-derived labels (LAION, OPENAI-1536):** Using vector-associated metadata (e.g., text or images), we follow the procedure of [46] to construct a candidate label set and then assign to each vector its top-3 most relevant labels.
- (3) **Synthetic labels (SIFT, MSMARCO, DEEP):** Since keywords and tags in real systems typically follow heavy-tailed distributions, prior work models label frequency using Zipf (power-law) distributions [6, 21]. We reuse publicly available code to generate labels for both base and query vectors, varying the alphabet size  $|\Sigma| \in \{8, 12, 24, 32\}$  [62]. In subsequent experiments, we also evaluate alternative label distributions (Uniform, Poisson, and Multinomial) as well as real datasets to assess robustness.

Table 3: The Statistics of Datasets

Dataset	Dimension	Size	Query Size	Type	Label
SIFT	128	1,000,000	10000	Image	Synthetic
MSMARCO	1024	1,000,000	1000	Text	Synthetic
LAION	512	1,182,243	1000	Image	Meta
OpenAI-1536	1536	999,000	1000	Text	Meta
PAPER	200	2,029,997	10000	Text	Real
TripClick	768	1,055,976	1000	Text	Real
Arxiv	4000	100,000	1000	Text	Real
DEEP	96	100,000,000	1000	Image	Synthetic

**Metrics.** The evaluation covers both 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 [6, 46]. All results are reported as averages. We also report the **average selectivity** of a query workload. For each query  $(q, L_q)$ , selectivity is defined as  $|S(L_q)|/|S|$ , the fraction of database vectors whose label sets contain  $L_q$ . We annotate the **average selectivity** (denoted *Ave. Select*) across all test queries at the top of each figure to highlight this value.

**Algorithms.** We compare the following algorithms:

- **ELI-0.2:** Our greedy method for the efficiency-constrained variant, where 0.2 denotes the elastic-factor threshold.
- **ELI-2.0:** Our method for the space-constrained variant, where 2.0 indicates the space bound (at most twice the original dataset size).
- **UNG:** Unified Navigating Graph for AKNN search [6].
- **ACORN-1:** ANN Constraint-Optimized Retrieval Network with low construction overhead [46].
- **ACORN- $\gamma$ :** ANN Constraint-Optimized Retrieval Network for high-efficiency search [46].
- **Brute-Force:** Scan all label-matched vectors to find the results.

**Implementation Details.** All code was implemented in C++ and compiled with GCC 9.4.0 using the `-Ofast` optimization flag. Experiments were conducted on a workstation equipped with Intel(R) Xeon(R) Platinum 8352V CPUs @ 2.10 GHz and 512 GB of memory. We employed multi-threading (144 threads) for index construction and a single thread for search evaluation. For the underlying index, we used HNSW [43] with parameters  $M = 16$  and  $efconstruct = 200$ . For ELI-0.2 ( $< 1.0$ ), we applied the index-selection method to achieve a fixed elastic factor of 0.2. For ELI-2.0 ( $> 1.0$ ), we used the fixed-space method, allowing at most double the original database size to maximize efficiency. For other baselines, such as ACORN and UNG, we adopted the default parameters from their respective papers:  $\alpha = 1.2$ ,  $L = 100$  for UNG;  $\gamma = 80$ ,  $M = 128$ ,  $M_B = 128$  for ACORN- $\gamma$  on the LAION and TripClick datasets; and  $\gamma = 12$ ,  $M = 32$ ,  $M_B = 64$  for the remaining datasets.

### 5.2 Experiment Results

**Exp-1: Comparison of Query Efficiency.** We begin by evaluating query efficiency (QPS) across various methods. Fig. 8 compares UNG, ACORN-1, ACORN- $\gamma$ , and our methods (ELI-0.2, ELI-2.0) under varying alphabet sizes  $|\Sigma|$  and label distributions. Points toward the upper right indicate better performance. Our methods achieve the best trade-off between search efficiency and accuracy. For example, at 95% recall with  $|\Sigma| = 12$ , ELI-0.2 delivers a 4 $\times$  speedup over

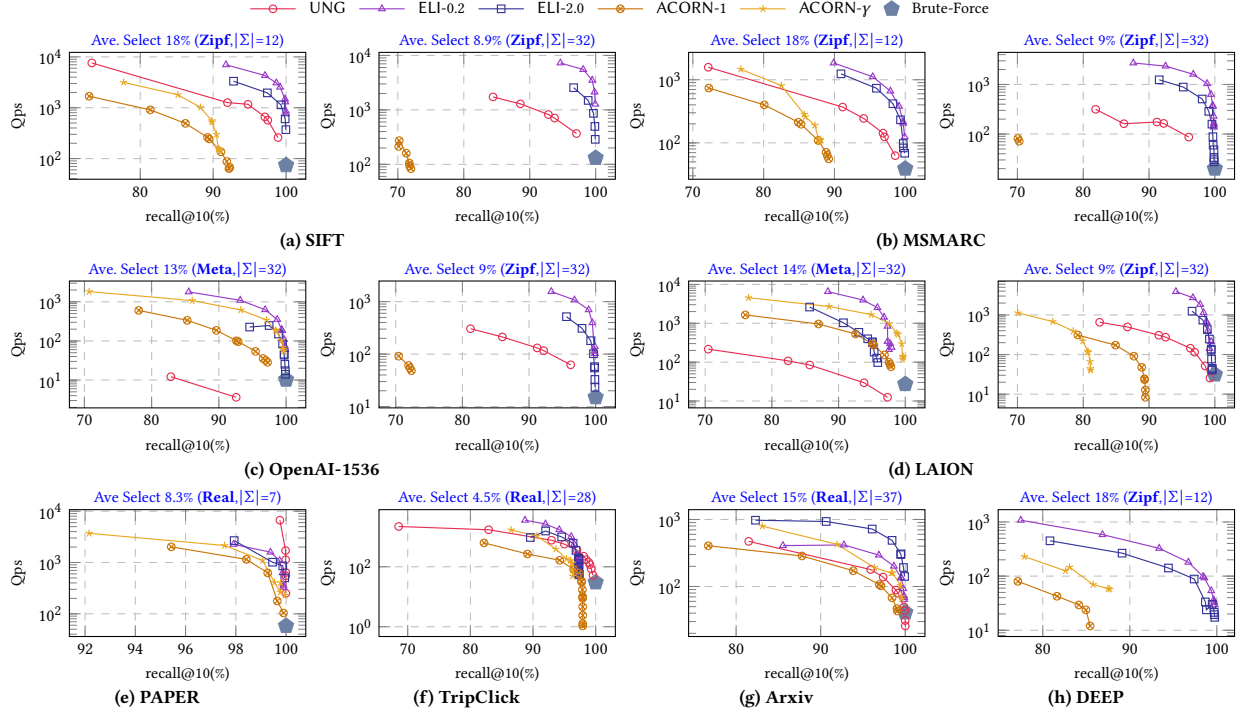


Figure 8: The Efficiency Comparison Across Methods

UNG on SIFT and a 10× speedup on MSMARCO. Performance remains stable across different  $|\Sigma|$  values and real label distributions, whereas UNG’s retrieval efficiency drops sharply as  $|\Sigma|$  grows.

Thanks to the elastic factor, ELI-0.2 sustains nearly 12× higher query speed than UNG when  $|\Sigma| = 32$  on the OpenAI dataset. In contrast, ACORN plateaus in accuracy for large  $|\Sigma|$  due to limitations in its PreFiltering strategy: as  $|\Sigma|$  increases, ACORN accuracy declines and UNG efficiency degrades, while ELI-0.2 maintains consistently high efficiency. Although larger  $|\Sigma|$  lowers average query selectivity, our design keeps runtime stable. A modest gap appears between ELI-0.2 and ELI-2.0: under a Zipf distribution, the elastic factor achievable with only double the space budget (ELI-2.0) is typically below 0.2 for large  $|\Sigma|$ , so ELI-0.2 often performs better because of its higher elastic-factor threshold.

**Exp-2: Effect of Query Selectivity.** To further compare methods, we vary query selectivity and measure its impact on query efficiency. We evaluate on Arxiv with four settings: *No Filter* (100% selectivity), 30%, 10%, and 5% selectivity, as shown in Fig. 9. In the No-Filter setting, the problem reduces to standard AKNN search without label constraints. Here, ELI-0.2 and ELI-2.0 match the performance of the state-of-the-art HNSW-PostFiltering method and outperform other baselines. At 30% selectivity, ELI-2.0 consistently surpasses UNG and ACORN across the full recall range, while PostFiltering performs nearly the same as ELI-0.2 as the top index satisfies  $c \geq 0.2$ . When selectivity drops to 10% and 5%, the advantage of our methods widens: they sustain high QPS even at high recall. These results validate the elastic-factor cost model: with sufficiently large  $c$  (e.g.,  $c \geq 0.2$ ), the additional computation scales as  $k/c$ . We also compare with brute-force search, which becomes competitive when very few vectors match the labels (e.g., fewer than 4K). In practice, we

Table 4: The Comparison of Indexing Time (s)

	UNG	ACORN-1	ACORN- $\gamma$	ELI-0.2	ELI-2.0
SIFT	405	7	62	34	25
MSMARCO	459	29	290	148	87
PAPER	65	18	128	39	64
LAION	27	105	3238	121	63
TripClick	283	137	4289	127	81
OpenAI-1536	218	47	568	409	166
Arxiv	116	9	50	23	25
DEEP	-	912	11328	8864	3991

Table 5: The Comparison of Index Size (Mb)

	Base	UNG	ACORN-1	ACORN- $\gamma$	ELI-0.2	ELI-2.0
SIFT	488	186	442	485	634	382
MSMARCO	3906	233	442	485	634	382
PAPER	1548	372	898	986	699	650
LAION	2309	204	2038	1820	871	405
TripClick	3093	192	1820	1625	814	540
OpenAI-1536	5853	261	442	485	782	342
Arxiv	1525	20	80	85	26	30
DEEP	36621	-	44262	48591	85293	37902

therefore build indexes only for query-label sets containing at least about 4K vectors and use brute force otherwise.

**Exp-3: Index Selection Statistics.** Recall that our methods aim to select a subset of indexes  $\mathcal{I}$  from the universal index set  $\mathcal{I}$ . To illustrate how the selection works, we report both the number of indexes chosen and the total number of vectors covered by the indexes selected by our methods (ELI-0.2 and ELI-2.0). We evaluate on real datasets (OPENAI, LAION, TripClick, Arxiv, and PAPER) and also vary the label distribution of PAPER, with results shown in Fig. 10. The top panel of Fig. 10 presents the total number of candidate indexes  $|\mathcal{I}|$  as well as the number selected by ELI-0.2 and ELI-2.0. The bottom panel shows the total size of all candidate indexes (i.e.,

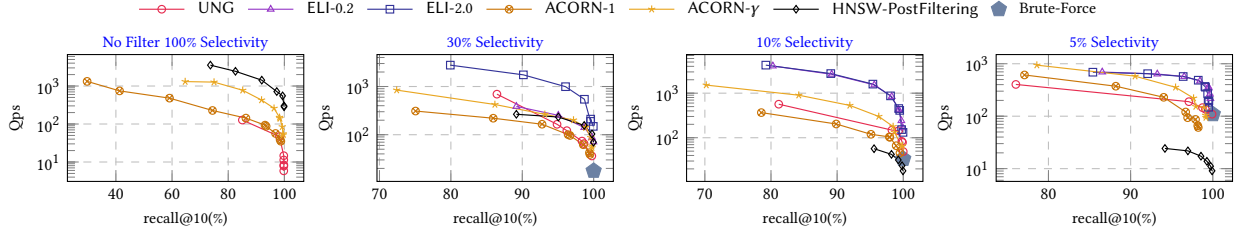


Figure 9: The Effect of the Query Selectivity (on Arxiv)

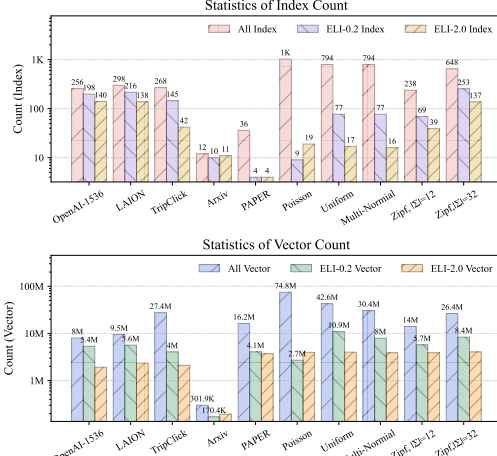


Figure 10: The Statistics of Index Selection by Our Methods

total number of vectors across indexes) and the size of the indexes actually selected by ELI-0.2 and ELI-2.0.

From Fig. 10, we observe: (1) **Compact label-set selection.** Both methods select up to an order of magnitude fewer indexes than the full candidate set, verifying the effectiveness of index sharing in reducing index count. (2) **Significant reduction in indexed size.** The bottom panel shows that ELI-0.2 indexes only a small fraction of the total base vectors, while ELI-2.0 indexes roughly twice the number of base vectors. (3) **Robustness across distributions.** Under various label distributions for PAPER (Uniform, Poisson, Multinomial, and Zipf), our methods always achieve substantial reductions in both the number of selected indexes and their total size.

**Exp-4: Indexing Time and Space.** We compare indexing time in Table 4 and index size in Table 5. For datasets with real labels, results are reported under their native label distributions. For synthetic datasets, we report results with  $|\Sigma| = 32$ , consistent with the efficiency comparison in Fig. 8. (1) **Indexing time.** Table 4 shows that our methods (especially ELI-2.0) are far more efficient than UNG and ACORN- $\gamma$ . Specifically, UNG requires nearly 20 $\times$  the indexing time on SIFT and 5 $\times$  on MSMARCO, while ACORN- $\gamma$  averages about 2 $\times$  the indexing time of our methods. Although ACORN-1 builds indexes quickly, its omission of label information results in poorer search efficiency. (2) **Index size.** Table 5 reports index sizes, where *Base* denotes the size of the raw vectors. On the OpenAI-1536 dataset, all methods require only about one-tenth of the total storage space. Our methods achieve the best search performance with comparable index size. In particular, both ACORN and ELI-2.0 use roughly twice the HNSW space of the original vectors, yet ELI-2.0 achieves strong search performance with much

smaller additional space. UNG saves the most space, but in high-dimensional settings its index size is only a small fraction of the base vector dataset, leading to lower retrieval efficiency.

**Exp-5: Varying Label Distributions.** Label distributions influence label-hybrid AKNN search, so we evaluate performance under different distributions—Zipf, Uniform, Poisson, and Multinomial—with  $|\Sigma| = 12$ . Experiments are conducted on the PAPER dataset, and results are shown in Fig. 11. (1) **Stable performance.** Our methods perform consistently across all tested distributions. In contrast, the search accuracy of other methods varies significantly with the distribution: under Uniform and Multinomial settings, ACORN fails to reach 80% recall, while UNG and our methods maintain high accuracy. (2) **High efficiency.** Across all distributions, our methods remain highly competitive. They achieve recall comparable to UNG at high thresholds and deliver 3 $\times$ –5 $\times$  better search efficiency.

**Exp-6: Index Update Evaluation.** We evaluate the effectiveness of our index-update strategy on the ARXIV and SIFT datasets. Each dataset is partitioned into two halves. The first 50% of the data, with label-set size  $|\Sigma| = 12$ , is used to construct the initial index set  $\mathcal{I}$ . The remaining 50%, featuring an expanded label set of  $|\Sigma| = 24$ , simulates the insertion of both new vectors (50% more vectors) and an enlarged query workload (12 additional labels).

**Query efficiency.** Fig. 12 reports query efficiency. ELI-0.2 and ELI-2.0 denote the methods rebuilt from scratch using the full dataset, whereas ELI'-0.2 and ELI'-2.0 represent our batch-update strategy, which first builds indexes on the initial 50% of data and then incrementally updates them. The results show that periodic updates incur less than a 5% drop in query efficiency compared with full re-computation, demonstrating that the lazy update mechanism preserves practical performance.

**Update cost.** Figure 13 compares update cost for ELI'-0.2 and ELI'-2.0 against rebuilding from scratch. The left panel shows update time: the rightmost bar corresponds to full re-computation on 100% of the data, while the three bars to the left represent incremental insertions of 10%, 20%, and 50% new vectors into the initial index built on the first 50% of data. Update time naturally increases with the number of inserted vectors but remains well below the cost of rebuilding, even when 50% of the data is added. The right panel reports space usage (in the unit of updated vector count). As more vectors are inserted, index size grows slightly, but remains smaller than that required for full re-computation. Notably, ELI-2.0 prunes redundant indexes and space when 50% of vectors are inserted, because the method enforces a fixed space budget.

Overall, these experiments show that our incremental update strategy achieves significantly lower update time and space overhead compared with rebuilding from scratch, while incurring only negligible loss in query efficiency (Fig. 12).



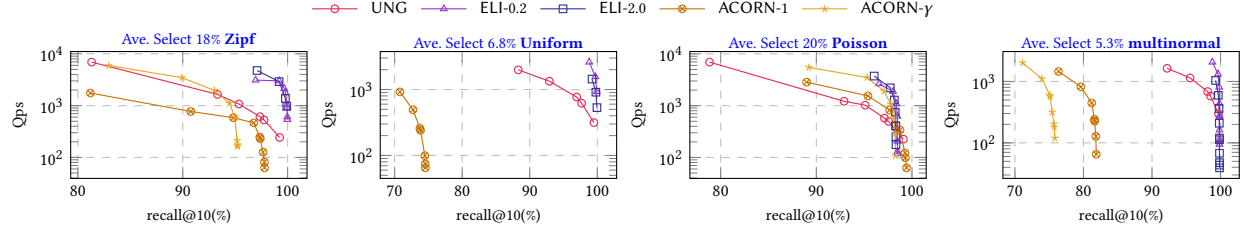


Figure 11: Varying Label Distributions

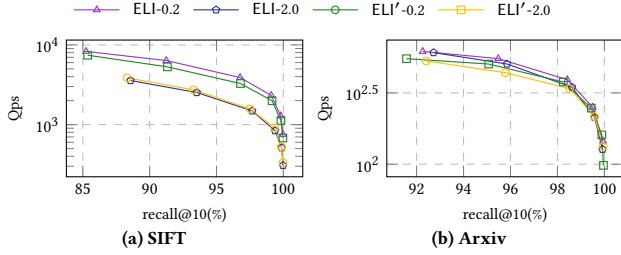


Figure 12: Query Performance Drop Caused by Incremental Updates

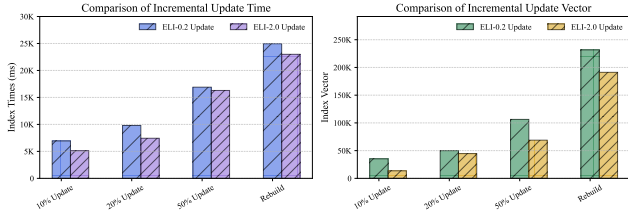


Figure 13: The Time and Space Cost of Incremental Updates

## 6 RELATED WORK

**Attribute-Hybrid AKNN Search.** When vectors carry label attributes (as defined in this paper) or numerical attributes (e.g., [58]), an **attribute-hybrid** AKNN search can be defined to return approximate  $k$ -nearest neighbors whose attributes satisfy given constraints. Such searches can be handled using either attribute-free or attribute-dependent solutions.

**Attribute-Free Solutions.** These methods ignore attributes during index construction and consider them only at query time. Specifically, the PreFiltering and PostFiltering search strategies are widely used, each performing differently under varying query workloads. Systems such as ADB [56], VBASE [65], CHASE [41], and Milvus [50] choose between these strategies using cost models. ACORN also follows the PreFiltering strategy but builds a denser graph index to mitigate its connectivity issues. Because these approaches ignore attributes during index construction, their search accuracy and efficiency lag behind methods that incorporate attributes directly.

**Attribute-Dependent Solutions.** These methods build indexes by considering the attribute type. (1) For attribute-hybrid AKNN search with *numerical* attributes, recent work [11, 58, 60] organizes query-specified numeric intervals using segment trees, while SeRF [68] further compresses indexes across overlapping intervals. (2) For attribute-hybrid AKNN search with *label* attributes, UNG [6] exploits label-set inclusion, adding cross-group edges so the graph index searches only filtered vectors. NHQ [51] incorporates attribute similarity into vector similarity, requiring manual tuning of vector and attribute weights. The recent DEG approach [63] addresses this by dynamically adjusting graph neighbors.

**Data Cube Computation.** Existing studies on OLAP and data-cube computation select group-by aggregates over the subset lattice of dimensions, exploiting the fact that coarser aggregates (supersets) can be derived from finer-grained materializations (subsets) [24, 27, 44]. Our setting shares this lattice structure and the reuse principle: an index built for a query-label set can serve queries with superset labels via label containment. However, key differences remain: (1) Cube computation relies on algebraic properties of aggregates (e.g., distributive or algebraic measures) to derive superset results exactly from subsets, whereas our reuse is containment-based, yielding a query cost bounded by  $O(C + k/e)$  on graph-based indexes, where  $e$  is the elastic factor and  $k/e$  is the additional cost. (2) Cube/view selection minimizes aggregate-query latency subject to storage limits, while our Elastic Index Selection (EIS) problem selects a subset of query-label sets to index so as to satisfy both space budgets and workload-wide elastic-factor coverage, ensuring predictable AKNN efficiency under label constraints. (3) Although both problems are NP-hard and typically approached with greedy heuristics, our benefit function and feasibility constraints are tailored to vector-search semantics and the label-containment relationship.

**Frequent Itemset Mining.** Our work is conceptually related to Frequent Itemset Mining (FIM), where labels correspond to items and label sets to transactions [25, 26, 64]. Both exploit the anti-monotone property of support (the Apriori principle), with classic algorithms such as Apriori and FP-Growth widely used for FIM [25, 26]. The objectives, however, differ fundamentally. FIM finds all itemsets whose support exceeds a threshold for pattern discovery, whereas our ELI task is an optimization problem: selecting a subset of query-label sets on which to build indexes. This selection is guided not by absolute support but by a novel *elastic factor* modeling query efficiency. Thus, while FIM outputs frequent patterns, our method produces a concrete indexing plan for AKNN search, making the formulation and goals distinct.

## 7 CONCLUSION

This paper investigates the label-hybrid AKNN search problem and shows that an index built for a query-label set remains effective for all its supersets with bounded query time. Building on this insight, we formalize the **Elastic Index Selection** (EIS) problem, prove that EIS is NP-complete, and introduce two optimization variants solved by efficient greedy algorithms. Extensive experiments demonstrate the effectiveness and superiority of our approach. As future work, we plan to explore hardware acceleration for further speedups.

## 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] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.
  - [3] Artem Babenko and Victor S. Lempitsky. 2015. The Inverted Multi-Index. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 6 (2015), 1247–1260.
  - [4] 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.
  - [5] 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.
  - [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] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*. 380–388.
  - [8] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.
  - [9] 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.
  - [10] 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.
  - [11] Joshua Engels, Benjamin Landrum, Shangdi Yu, Laxman Dhulipala, and Julian Shun. 2024. Approximate Nearest Neighbor Search with Window Filters. *ICML 2024* (2024).
  - [12] 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.
  - [13] 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.
  - [14] 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.
  - [15] Jianyang Gao, Yutong Gou, Yuxuan 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).
  - [16] 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>
  - [17] 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.
  - [18] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 4 (2014), 744–755.
  - [19] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.
  - [20] 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.
  - [21] 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.
  - [22] 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>
  - [23] Yutong Gou, Jianyang Gao, Yuxuan 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.
  - [24] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery* 1, 1 (1997), 29–53.
  - [25] Jiawei Han, Jian Pei, and Hanghang Tong. 2022. *Data mining: concepts and techniques*. Morgan kaufmann.
  - [26] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM sigmod record* 29, 2 (2000), 1–12.
  - [27] Venky Harinarayan, Anand Rajaraman, and Jeffrey D Ullman. 1996. Implementing data cubes efficiently. *Acm Sigmod Record* 25, 2 (1996), 205–216.
  - [28] Hazar Harmouch and Felix Naumann. 2017. Cardinality estimation: An experimental survey. *Proceedings of the VLDB Endowment* 11, 4 (2017), 499–512.
  - [29] Ben Harwood and Tom Drummond. 2016. Fannq: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.
  - [30] 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.
  - [31] Patrick Iff, Paul Bruegger, Marcin Chrapek, Maciej Besta, and Torsten Hoeftler. 2025. Benchmarking Filtered Approximate Nearest Neighbor Search Algorithms on Transformer-based Embedding Vectors. [arXiv:2507.21989 \[cs.DB\]](https://arxiv.org/abs/2507.21989) <https://arxiv.org/abs/2507.21989>
  - [32] 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.
  - [33] 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.
  - [34] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).
  - [35] 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.
  - [36] 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.
  - [37] 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.
  - [38] 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.
  - [39] 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.
  - [40] 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.
  - [41] Rui Ma, Kai Zhang, Zhenying He, Yanan 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).
  - [42] Yu A Malkov and Dmitry A Yashunin. [n.d.]. [hnswlib](https://github.com/nmslib/hnswlib). <https://github.com/nmslib/hnswlib>.
  - [43] 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.
  - [44] Imene Mami and Zohra Bellahsene. 2012. A survey of view selection methods. *Acm Sigmod Record* 41, 1 (2012), 20–29.
  - [45] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11 (2002), 28–46.
  - [46] 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.
  - [47] 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>
  - [48] 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.
  - [49] 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.
  - [50] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xianguy 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.
  - [51] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jionggang Ni. 2022. Navigable proximity graph-driven native hybrid queries with structured and unstructured constraints. *arXiv preprint arXiv:2203.13601* (2022).
  - [52] 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).

Notation	Description
$S$	Set of vectors, each with an associated label set
$L_i, L_q$	Base and query label sets
$S(L_q)$	Subset of $S$ whose label sets contain $L_q$
$ L $	Cardinality of label set $L$
$\mathcal{L}$	Collection of all label sets
$ \mathcal{L} $	Number of label sets in $\mathcal{L}$
$\mathcal{I}$	Collection of indexes
$\mathbb{L}$	The selected indexes from $\mathcal{L}$
$\Sigma$	Label alphabet (all possible labels)
$N$	Cardinality of $S$
$q$	Query vector
$e$	Elastic factor
$d$	Dimensionality of vectors in $S$
$\delta(u, v)$	Distance (or similarity) between $u$ and $v$
$I$	AKNN search index (e.g., HNSW)
$\tau$	Space budget
$c$	Elastic-factor threshold

- [53] 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>
- [54] 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.
- [55] 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.
- [56] 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.
- [57] 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.
- [58] Yuxuan 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).
- [59] 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).
- [60] 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).
- [61] 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).
- [62] Mingyu Yang, Wenxuan Xia, Wentao Li, Raymond Chi-Wing Wong, and Wei Wang. 2025. *Technical Report*. <https://github.com/mingyu-hkustgz/LabelANN>
- [63] 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.
- [64] Mohammed Javeed Zaki. 2002. Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering* 12, 3 (2002), 372–390.
- [65] 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.
- [66] 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.
- [67] 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).
- [68] 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.

## A APPENDIX

### A.1 Common Used Notations

We list the commonly used notations in the following table.

### A.2 Proofs

**Proof of Lemma 3.3.** We analyze the expected search time of the PostFiltering strategy on a graph index by decomposing it into two phases: (1) the initial search to locate the first nearest neighbor (top-1 result), and (2) iterative expansion to collect  $k$  neighbors satisfying the query’s label constraint.

*Phase 1: Initial Search* Let  $C$  denote the expected cost of finding the top-1 neighbor of query  $q$  on the underlying graph index. This abstracts the entry cost of popular structures such as HNSW or NSG, typically analyzed as  $O(\log N)$  under standard assumptions [13, 46], where  $N$  is the number of indexed points.

*Phase 2: Iterative Expansion* After the first match, PostFiltering inspects candidates in increasing distance order until  $k$  items satisfying  $L_q$  are found. Efficiency depends on the probability that a candidate satisfies the label constraint. We assume (i) label attributes are independent of vector positions, and (ii) label-matched points are uniformly distributed in the space. If the elastic factor satisfies  $e(S(L_q), \mathbb{L}) \geq c$ , then at least a  $c$ -fraction of candidates within the selected index set  $\mathbb{L}$  match  $L_q$ . Under these assumptions, the probability that any visited candidate matches  $L_q$  is  $p \geq c$ . Finding  $k$  matches is therefore a sequence of Bernoulli trials with success probability  $p$ , so the number of trials  $T$  follows a negative-binomial distribution with expectation  $\mathbb{E}[T] = k/p \leq k/c$ . Modern graph-based indexes [13, 23, 34, 42] retrieve the next neighbor in  $O(1)$  amortized time because each node maintains a bounded-degree neighbor list. Hence the expected cost of this phase is  $O(k/c)$ .

*Total Cost* Summing both phases, the expected time to obtain the top- $k$  label-hybrid AKNN results is  $O\left(C + \frac{k}{c}\right)$ . This matches and generalizes prior analyses of post-filtering strategies, such as HNSW-based filtered search [46], within our elastic-factor framework. As with HNSW and NSG, the bound captures *expected* complexity and does not guarantee worst-case recall, but it aligns with recent NSG analyses [60] that report the same  $O(C + k/c)$  cost under similar conditions.

### A.3 EXTRA DISCUSSIONS

**Other Types of Label Constraints.** Our work primarily addresses the *label-containment* constraint, where a vector’s label set must contain the query label set. Other label constraints can also be supported:

(1) *Label equality.* Here we require base vectors whose label set  $L_i$  exactly matches the query label set  $L_q$ . To handle AKNN search under this constraint, we group base vectors by their label sets to form a label-set inverted list. An index is then built for each group, and the overall dataset cardinality remains unchanged. For example, an index for  $L_q = \{A\}$  is built over base vectors  $x_2$  and  $x_5$ , both labeled  $\{A\}$ . When using a graph-based index—the most efficient



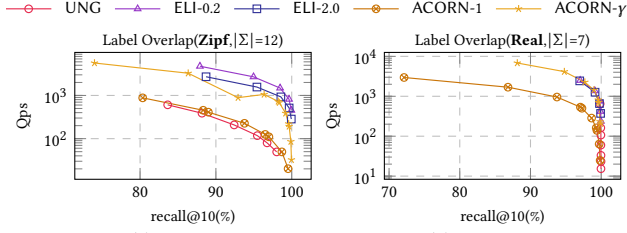


Figure 14: The Test of Label-Overlap Constraints

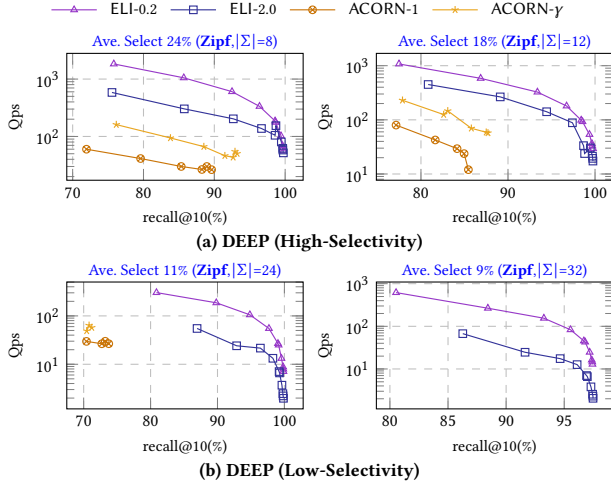


Figure 15: The Test of Scalability

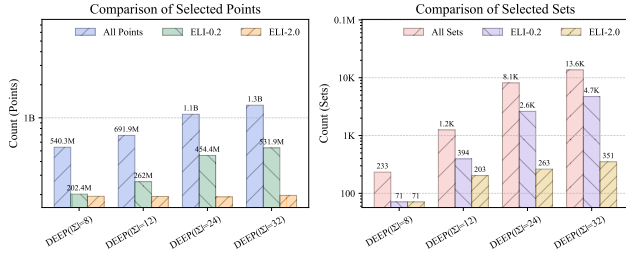


Figure 16: The Index Select Statistic of Large-Scale Data

structure to date—the total space complexity is  $O(NM)$ , where  $M$  is the maximum node degree constrained by edge occlusion.

(2) *Label overlap*. Here we require that a vector’s label set intersects the query label set, i.e.,  $L_i \cap L_q \neq \emptyset$ . To support AKNN search under this type of label constraint, we decompose the query into  $|L_q|$  subqueries, each with a single-label containment constraint. For example, a label-overlap query with label set  $L_q = \{AB\}$  can be answered by merging the results of label-containment queries with label sets  $L_q = \{A\}$  and  $L_q = \{B\}$  and taking their union.

These extensions show that our approach naturally generalizes beyond simple label containment, while we focus on the containment case throughout this paper.

**Vector Similarity Search.** Vector similarity search is closely related to AKNN search and has been extensively studied. Most prior work targets approximate search [19], since exact search is prohibitively expensive in high-dimensional spaces [32]. Among these methods,  $c$ -approximate nearest neighbor search returns results

within a factor  $c$  of the true distance and can be solved in sublinear time using locality-sensitive hashing (LSH) [9, 14, 19, 20, 30, 49, 66]. For the AKNN search problem in this paper, graph-based vector indexes [12, 13, 23, 34, 40, 43, 47, 53] represent the current state of the art and are significantly more efficient than LSH according to benchmarks [39]. In addition, inverted-list indexes [3, 35] and quantization-based methods [15, 17, 18, 22, 35, 36, 61] are widely used for AKNN search. Inverted indexes provide low space overhead, while quantization techniques speed up distance computations [52, 59]. In this paper, we employ the widely adopted HNSW algorithm [42] as the underlying index for each vector group, although other optimized AKNN libraries [34, 67] can be used as drop-in replacements.

## A.4 Extra Experiments

**Exp-7: Test of Label-Overlap Constraints.** We previously discussed how our method for AKNN search with label-containment constraints can be extended to handle label-overlap constraints. Specifically, a query with label overlap is decomposed into multiple single-label containment sub-queries; We solve each sub-query independently and then merge the results. We evaluate this approach on both synthetic SIFT data with a Zipf distribution and the real-world PAPER dataset, as shown in Fig. 14. The figure shows that, despite requiring multiple search calls for single-label containment sub-queries, our methods ELI-0.2 and ELI-2.0 achieve search performance comparable to ACORN- $\gamma$ , while delivering  $2\times$ – $10\times$  the efficiency of ACORN and UNG. Since our current framework does not treat overlap queries as part of the optimization workload, we plan to explore extensions of our greedy method to support such queries and, more broadly, other complex label constraints.

**Exp-8: Scalability to Large Datasets.** To validate the scalability of our algorithm on large-scale data, we adopt the DEEP dataset containing 100M vectors of 96 dimensions. Labels follow a Zipf distribution with varying query selectivities, and the results are shown in Fig. 15. Because brute-force search achieves only 0.2–0.6 QPS, we omit it from the figure. Moreover, UNG fails to scale, encountering a core dump during index construction on the DEEP100M dataset. From Fig. 15, ACORN exhibits  $4\times$  worse performance than our ELI-0.2 method at 90% recall and  $3\times$  worse than ELI-2.0. Our methods maintain consistent performance as the alphabet size  $|\Sigma|$  increases, whereas ACORN fails to reach 90% recall for large  $|\Sigma|$ .

We also analyze index-selection statistics of our methods on this large graph by varying the alphabet size, with results in Fig. 16. ELI-2.0 indexes only an additional 100M vectors with 351 selected index sets—just 1/12 of all possible vectors, excluding the base-vector index of 100M. ELI-0.2 selects more indexes, indexing roughly 400M vectors (about one-third of the 1.2B candidates) across 4.7K sets. This larger index set pays off: ELI-0.2 achieves  $4\times$ – $6\times$  higher query efficiency than ELI-2.0. These results highlight the trade-off between index size and query efficiency in real-world large-scale scenarios for our ELI-2.0 and ELI-0.2 methods.

**Exp-9: Varying Alphabet Size.** In Exp-1 we varied the alphabet size over a limited range; here we further test the search efficiency of different methods—particularly UNG, ACORN, and our ELI-0.2—under much larger alphabets. We increase  $|\Sigma|$  from 64 to 128, 256,

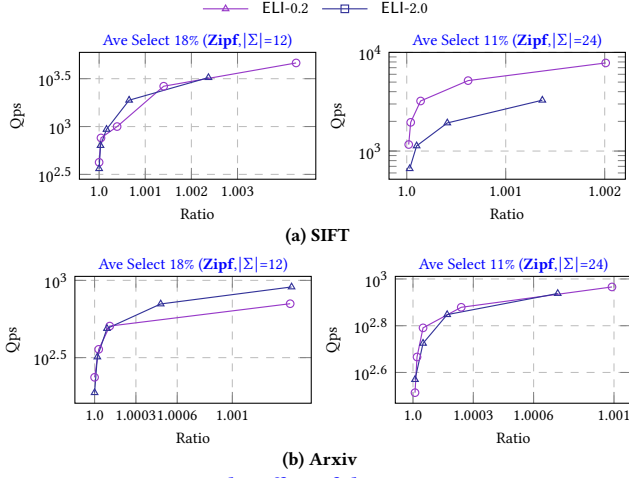


Figure 20: The Effect of the Distance Ratio

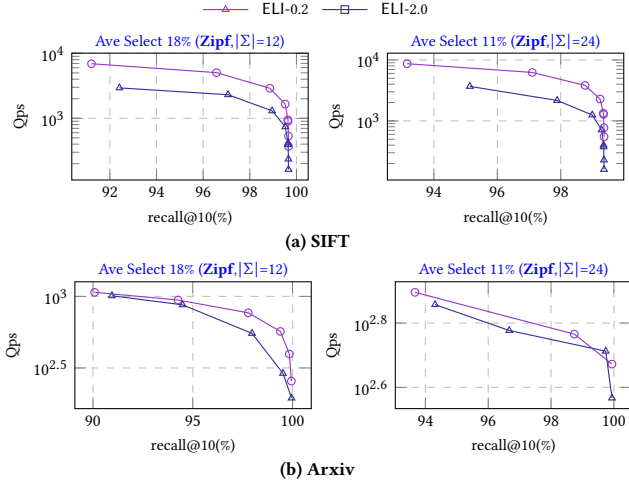


Figure 21: The Test of Maximum Inner Product Search

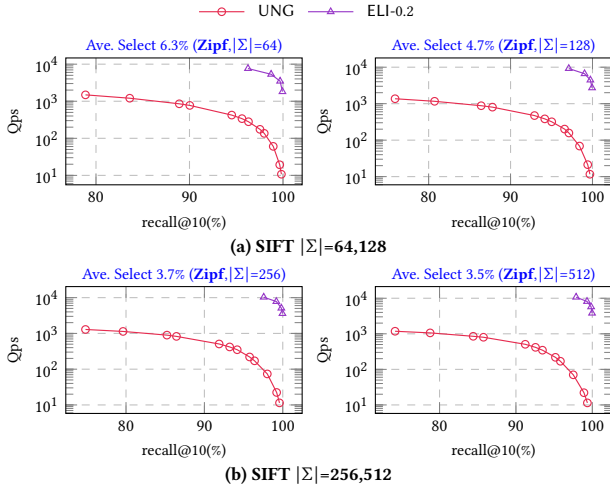


Figure 17: Varying the Alphabet Size  $|\Sigma|$

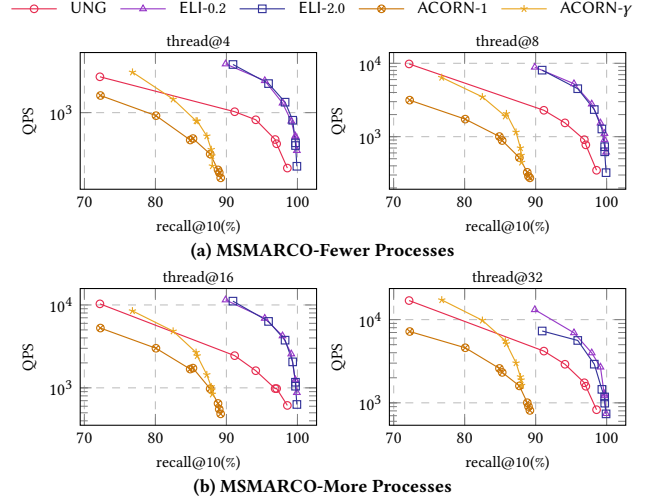


Figure 18: Varying the Thread Number (on MSMARCO)

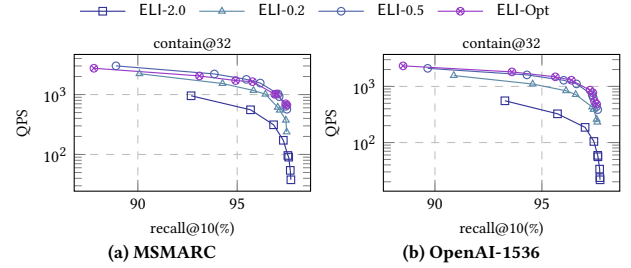


Figure 19: The Comparison with the Exhaustive Method

and 512 to evaluate UNG and ELI-0.2. The ACORN method fails to achieve 80% recall at these scales and is therefore omitted from Fig. 17. Results show that our method achieves nearly an 800 $\times$  speedup in search efficiency at 99% recall when  $|\Sigma| = 512$ . This improvement arises because the fixed-efficiency design of ELI-0.2 exploits the lower selectivity of large alphabets. In contrast, UNG suffers substantial degradation in search efficiency as  $|\Sigma|$  grows. We also compare index-build times. For  $|\Sigma| = 512$ , ELI-0.2 builds the index in 155 seconds, whereas UNG requires 2,091 seconds—over 13 $\times$  longer—demonstrating the superior scalability of ELI-0.2.

**Exp-10: Varying Thread Number.** During search, we apply parallelism to improve performance. To evaluate scalability, we vary the number of threads from 4 to 32 with  $|\Sigma| = 12$  and report results on the MSMARCO dataset (other datasets show similar trends). Figure 18 presents the results. Our methods achieve the highest search performance across all thread settings, demonstrating robustness in multi-threaded environments. Moreover, our architecture invokes only a single sub-index for each query, making it well suited for distributed systems—a direction we plan to explore in future work.

**Exp-11: Comparison with the Exhaustive Method.** Because our method selects a subset of indexes from the universal set corresponding to the query workload, we compare it with an *exhaustive* method that builds indexes for *all* possible query-label sets (i.e., selects the entire universal index set). We also introduce ELI-0.5, an efficiency-constrained variant of EIS with the elastic-factor threshold increased to 0.5. Figure 19 shows the search efficiency of ELI-2.0, ELI-0.2, ELI-0.5, and the exhaustive method (ELI-Opt) under

a Zipf distribution with  $|\Sigma| = 32$  on the MSMARCO and OpenAI-1536 datasets. The results indicate that ELI-0.2 and ELI-0.5 achieve near-optimal search efficiency at high recall, closely matching ELI-Opt. ELI-2.0 delivers roughly  $3\times$  lower QPS than the exhaustive method but requires far less space. In terms of index size, ELI-2.0, ELI-0.2, ELI-0.5, and ELI-Opt consume 383 MB, 634 MB, 812 MB, and 1,280 MB, respectively. Excluding the mandatory 192 MB top index, ELI-2.0 uses only about one-sixth of the space of the exhaustive approach while maintaining acceptable search performance. Moreover, ELI-0.5 requires only about half the space required by ELI-Opt to achieve nearly identical search efficiency, highlighting the advantage of our methods in efficiency-oriented scenarios.

**Exp-12: Effect of Distance Ratio.** To further assess search performance, we measure the *distance ratio* [7, 14, 66]: the ratio of the average distance between a query and the returned objects to the average distance between the query and its ground-truth  $k$  nearest neighbors. A ratio of 1.0 indicates that the exact  $k$ NN results are found; larger ratios allow more flexibility in the returned objects. We report QPS at varying distance ratios on the SIFT and

Arxiv datasets, with results shown in Fig. 20. Our methods require more time (i.e., achieve lower QPS) to reach very small distance ratios, but performance improves as the ratio increases. Notably, our methods often achieve an average distance ratio near 1.001, substantially outperforming prior LSH-based approaches [9, 14, 30, 49, 66], demonstrating our superiority from another perspective.

**Exp-13: Test of Maximum Inner Product Search.** Our default experiments use Euclidean distance to measure similarity between queries and base vectors. Here we replace Euclidean distance with the inner product and evaluate whether our method still performs well. This setting corresponds to Maximum Inner Product Search (MIPS). We test this extension on the SIFT and Arxiv datasets, with results shown in Fig. 21. As illustrated, MIPS delivers computational performance comparable to the Euclidean metric under our framework, maintaining high QPS even at high recall. These findings indicate that our method is broadly applicable across common vector similarity measures and align with prior evidence [2, 23] that graph-based indexes (e.g., HNSW) sustain high search efficiency across different similarity metrics.