Elastic Index Selection for Label-Hybrid AKNN Search

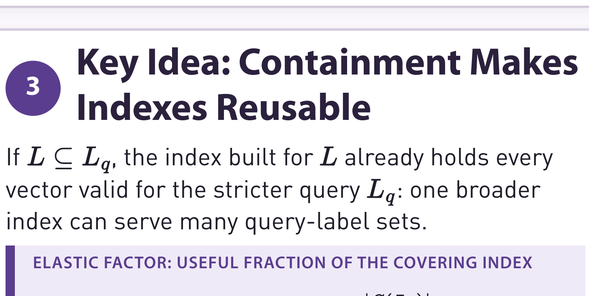
For label-hybrid approximate k-nearest neighbour (AKNN) search, Elastic Index Selection reuses indexes through label containment. By selecting a compact set of reusable indexes, it meets either an elastic-factor target or a fixed space budget.

Mingyu Yang · Wenxuan Xia · Wentao Li · Raymond Chi-Wing Wong · Wei Wang* * Corresponding author HKUST(GZ) · HKUST · University of Leicester

Paper

1 Problem: Label-Hybrid AKNN Search

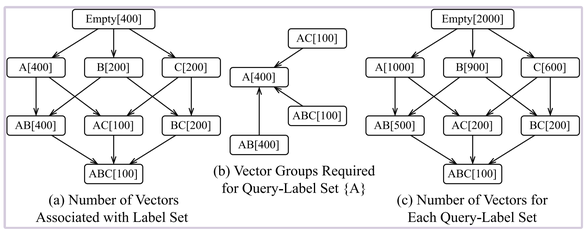
A query (q, L_q) asks for the approximate top-k nearest vectors whose label sets contain L_q.



Every result must satisfy $L_q \subseteq L_i$.

2 Motivation: Dedicated Indexes Explode Storage

A dedicated index per query-label set copies each vector into every index whose label set it contains.



Three labels already need $2.75\times$ the base entries.

3 Key Idea: Containment Makes Indexes Reusable

If $L \subseteq L_q$, the index built for L already holds every vector valid for the stricter query L_q: one broader index can serve many query-label sets.

ELASTIC FACTOR: USEFUL FRACTION OF THE COVERING INDEX

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

Keeping $e \geq c$ bounds the expected post-filtering time by $O(C + k/c)$ (Lemma 1). Lower c shares more; $c = 1$ degenerates to dedicated indexes.

4 Problem Formulation: Elastic Index Selection (EIS)

Candidates: one index $I_i = S(L_i)$ of cost $|I_i|$ per workload label set $L_i \in \mathcal{L}$; they form the universal set $\mathcal{I} = \{I_1, \ldots, I_n\}$, from which EIS selects a subset $\mathbb{I} \subseteq \mathcal{I}$:

EFFICIENCY-CONSTRAINED: GIVEN TARGET c

$$\min_{\mathbb{I}} \sum_{I_i \in \mathbb{I}} |I_i| \quad \text{s.t.} \quad e(S(L_i), \mathbb{I}) \geq c \ \ \forall L_i \in \mathcal{L}$$

SPACE-CONSTRAINED: GIVEN BUDGET τ

$$\max_{\mathbb{I}} \min_{L_i \in \mathcal{L}} e(S(L_i), \mathbb{I}) \quad \text{s.t.} \quad \sum_{I_i \in \mathbb{I}} |I_i| \leq \tau$$

EIS is NP-complete (reduction from 3-Set Cover), so both variants are solved approximately.

5 Method: Greedy Selection, Extended by Binary Search

Efficiency-constrained (given c): greedy selection.

1. Initialize $\mathbb{I} = \{I_\top\}$, the top index on all base vectors: every query is covered.
2. Iteratively add the candidate $I \in \mathcal{I} \setminus \mathbb{I}$ with the largest benefit $B(I, \mathbb{I})$.
3. Stop when $e(S(L_i), \mathbb{I}) \geq c$ for every $L_i \in \mathcal{L}$; return $\mathbb{I}$.

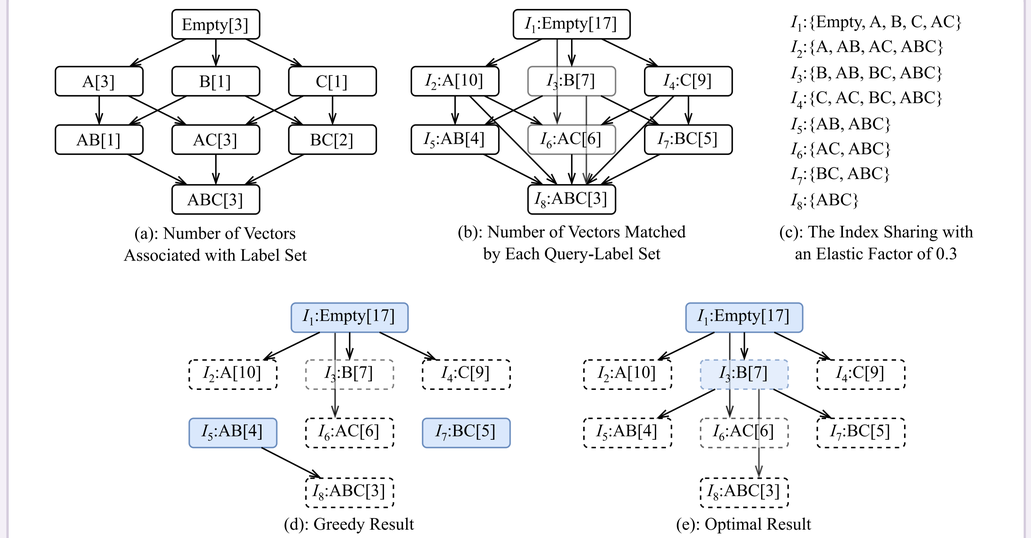
BENEFIT OF CANDIDATE I: COVERAGE NEWLY GAINED PER INDEXED VECTOR

$$B(I, \mathbb{I}) = \sum_{I_i \in \mathbb{C}'} \frac{|I_i|}{|I|}, \qquad \mathbb{C}' = \{\text{newly covered } I_i : \ I_i \subseteq I, \ |I_i|/|I| \geq c\}$$

Space-constrained (given τ): binary search over c, reusing the greedy. If threshold c is feasible within τ, so is any $c' < c$; binary search on $c \in (0, 1]$ runs the greedy per probe and keeps the largest feasible c, adding under 1% of the indexing time.

AT QUERY TIME: ROUTE TO THE COVERING INDEX WITH MAXIMUM ELASTIC FACTOR

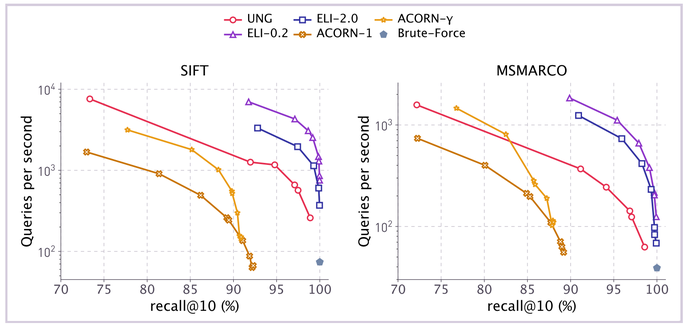
$$I^*(q) = \arg \max_{S(L_q) \subseteq I_i, \ I_i \in \mathbb{I}} e(S(L_q), I_i)$$



Running example, $c = 0.3$. (a) base vectors; (b) matching vectors per query; (c) service at $e \geq 0.3$; (d) the greedy selects I_1, I_5, I_7 at total cost 26; (e) the optimum selects I_1, I_3 at total cost 24.

6 Results: Query Efficiency

ELI-0.2 fixes target $c = 0.2$; **ELI-2.0** caps space at $2\times$ the base data. Zipf labels, $|\Sigma| = 12$.



At 95% recall: $4\times$ over UNG on SIFT and $10\times$ on MSMARCO; nearly $12\times$ over UNG on OpenAI-1536 at $|\Sigma| = 32$.

7 Results: Indexing Time and Space

Paper Tables 5–6, Zipf $|\Sigma| = 32$. Raw vectors: SIFT 488 MB, MSMARCO 3,906 MB.

Method	SIFT build (s)	SIFT size (MB)	MSMARCO build (s)	MSMARCO size (MB)
UNG	405	186	459	233
ACORN-γ	62	485	290	485
ELI-0.2	**34**	**634**	**148**	**634**
ELI-2.0	**25**	**382**	**87**	**382**

- ELI-2.0 reaches near-optimal query efficiency with a space overhead of 100%.
- UNG's index is smallest but slower in high dimensions; it cannot build DEEP (100M vectors).

8 Takeaway

- **Reuse instead of replicate:** one broader index serves every contained query-label set.
- **Speedups grow with the alphabet:** $10\times$ to $800\times$ over graph baselines at matched recall, up to $|\Sigma| = 512$.
- **Updates stay cheap:** batch updates keep query efficiency within 5% of a full rebuild (SIFT, Arxiv).

Boundary: with fully disjoint workload label sets there is no containment to exploit.

PVLDB 19(4): 617–630 · doi:10.14778/3785297.3785304

Artifact: github.com/mingyu-hkustgz/LabelANN · Presenter: wxia248@connect.hkust-gz.edu.cn