DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search

Jiuqi Wei*, Botao Peng*, Xiaodong Lee*, Themis Palpanas+

*: Institute of Computing Technology, Chinese Academy of Sciences

+: LIPADE, Université Paris Cité

VLDB2024 GUANGZHOU

Université Paris Cité

中国科学院计算技术研究所 INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

Background

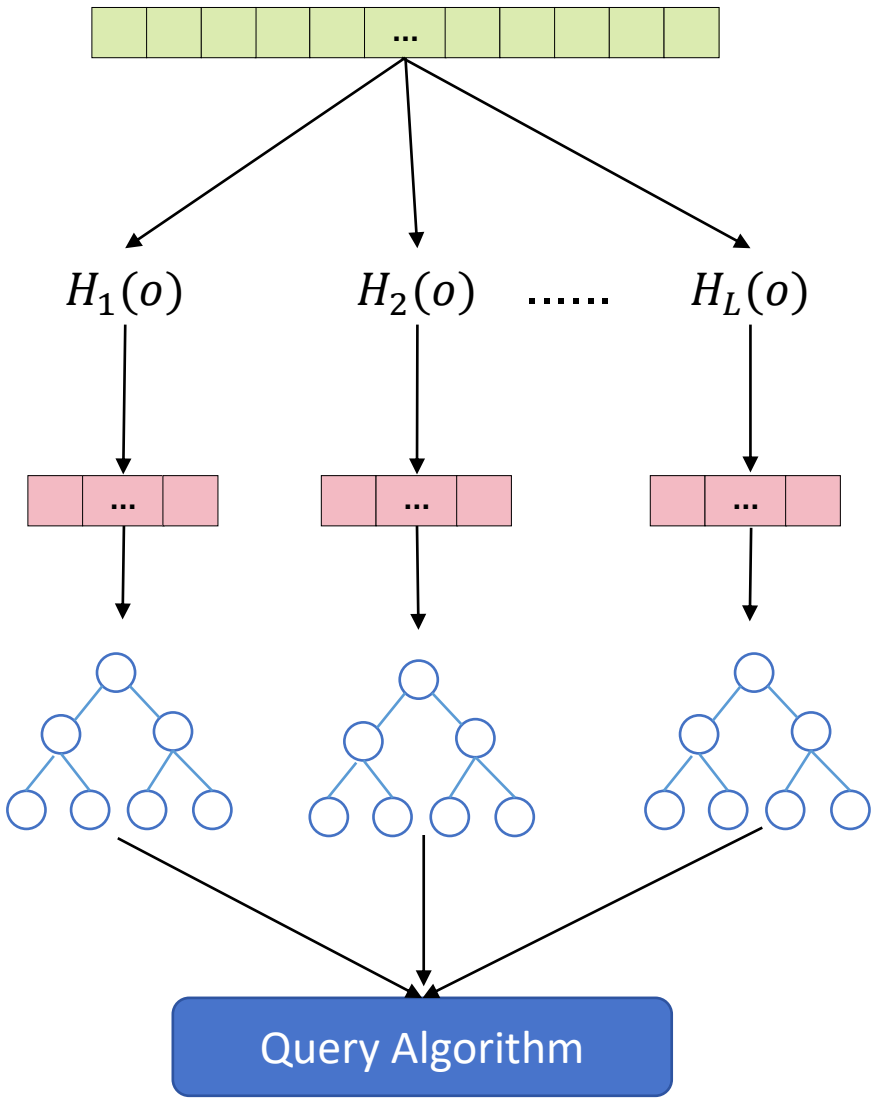
- Nearest neighbor (NN) search in high-dimensional Euclidean spaces is a fundamental problem in various fields.
  - database, information retrieval, data mining, machine learning,...
- However, NN search in high-dimensional datasets is challenging due to the “curse of dimensionality” phenomenon.
- In practice, Approximate NN (ANN) search is often used as an alternative, sacrificing some query accuracy to achieve a huge improvement in efficiency.

The core idea of LSH-based methods for ANN search

$d - dimension$

$H_i(o) = (h_{i1}(o), h_{i2}(o), \ldots h_{iK}(o))$

$K - dimension$



High-dimensional original space

Locality-Sensitive Hashing functions
$(L * K)$

Low-dimensional projected spaces
(L)
Indexing phase
(Construct L independent trees)

Query phase
(Search in L projected spaces)

Mainstream LSH-based Methods

- **Boundary constraint based methods (BC)**
  - **Core idea:** Each point is assigned into a hash bucket in the projected space, whose boundary is constrained by a **K-dimensional hypercube**. Among **L hash tables**, the point and query can be considered colliding as long as they are assigned to the same hash bucket **at least once**.
  - **Representative work**: DB-LSH (ICDE 22, **SOTA**)
- **Collision counting based methods (C2)**
  - **Core idea:** Construct L independent low-dimensional hash tables, and selects candidate points whose **number of collisions** with query is greater than a **threshold** t (t<L).
  - **Representative work**: R2LSH(ICDE 20) , VHP(VLDB 20)
- **Distance metric based methods (DM)**
  - **Core idea:** Select candidate points based on **distances to query** in the projected space.
  - **Representative work**: PM-LSH(VLDB 20)

Motivation

- Previous LSH-based methods mainly focus on designing query strategies, but pay little attention to the index structure.
- However, the index structure has a great impact on the performance of indexing and querying.
- It is necessary to comprehensively consider the index structure and query strategies to support efficient index construction and query answering.

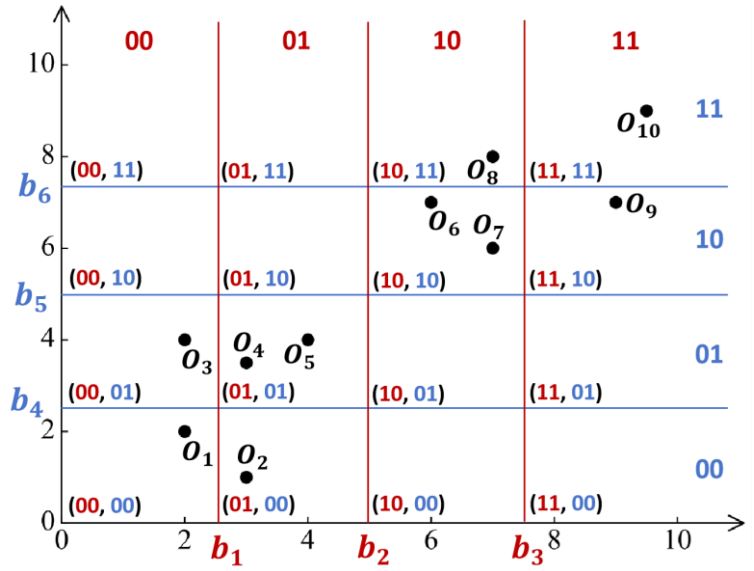
Can we design a novel tree structure and a novel LSH scheme that can be well adapted to support efficient and accurate ANN search?

Main Contributions

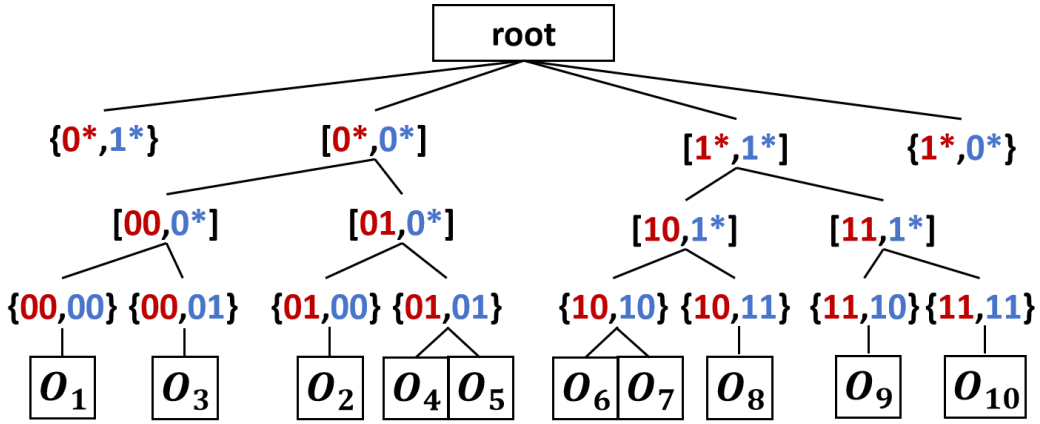
- We present a novel encoding-based tree structure called **Dynamic Encoding Tree (DE-Tree)**. DE-Tree has excellent indexing efficiency and can support efficient range queries.
- We propose **DET-LSH, a novel LSH scheme** based on DE-Tree. We provide a theoretical analysis showing that DET-LSH answers a c^2-k-ANN query with a constant success probability.
- We conduct extensive experiments, demonstrating that DET-LSH can achieve better efficiency and accuracy than existing LSH-based methods. While achieving better query accuracy, DET-LSH achieves up to **6x speedup in indexing time and 2x speedup in query time** over the SOTA LSH-based methods.

Our Method: Dynamic Encoding Tree (DE-Tree)

- **Dynamic breakpoints selection**: use *QuickSelect* algorithm and *divide-and-conquer* strategy to select breakpoints based on the data distribution.
- **Encoding scheme**: encode points into iSAX representations (256 symbols, 8-bit alphabet) based on the breakpoints.
- **Tree construction**: leaf nodes contain information about points, while internal nodes only contain index information.

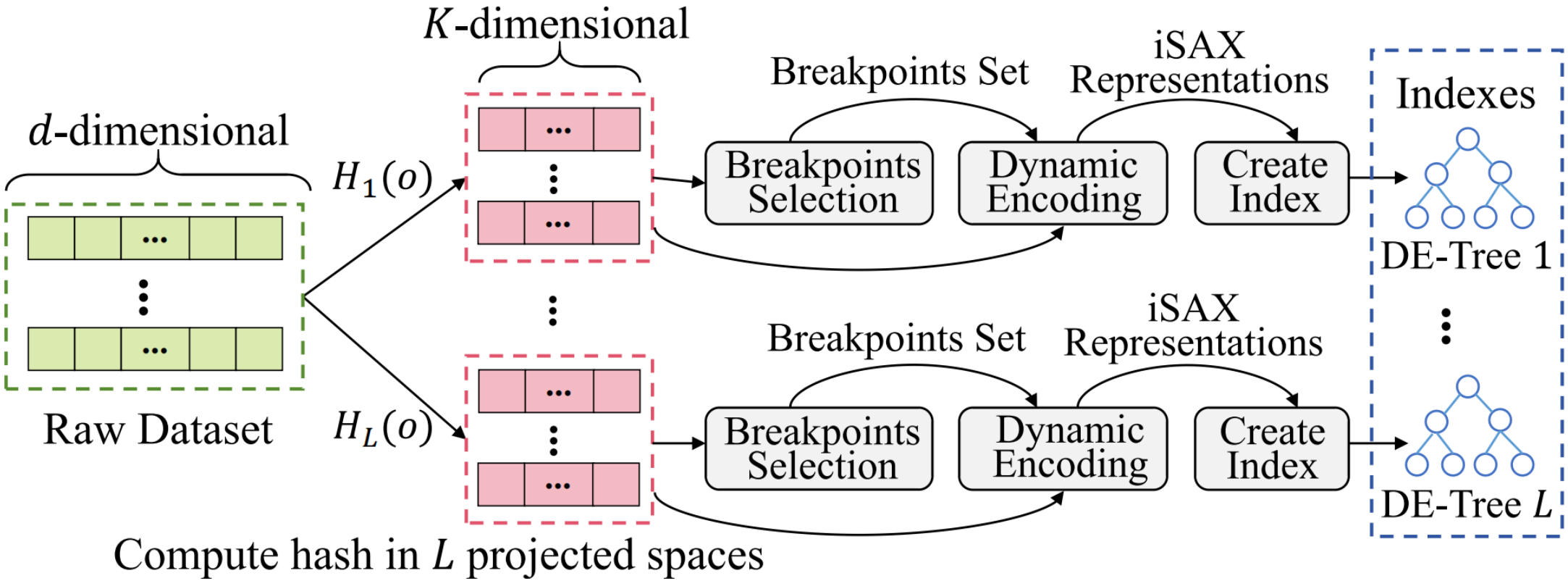


(a) Encode data points into iSAX representations.

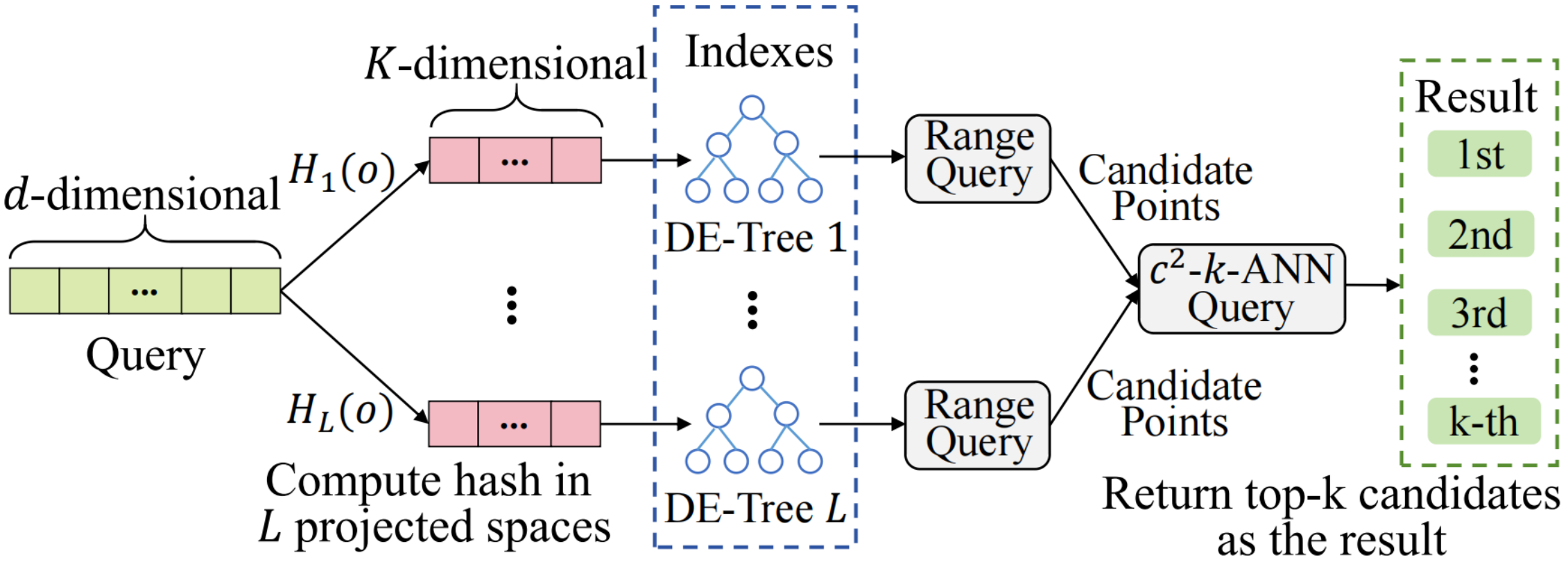


(b) An index based on the iSAX representations.

Our Method: DET-LSH Overview (Encoding + Indexing)



Our Method: DET-LSH Overview (Query)



Our Method: DET-LSH

- **Encoding phase**
  - 1. Generate $L * K$ hash functions. ($L = 4, K = 16$)
  - 2. Project all data points into 4 indepentent 16-dimensional spaces.
  - 3. Calculate the “breakpoints” of each project space.
  - 4. Encode all data points in 16-dimensional projected spaces based on corresponding breakpoints.

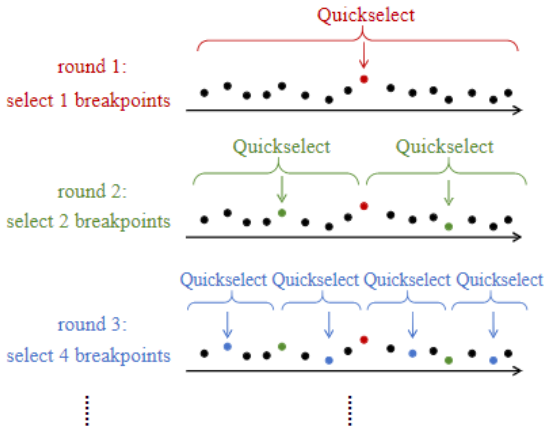


Figure 3: An illustration of Algorithm 1: select breakpoints in multiple rounds by *Quickselect* algorithm with *divide-and-conquer* strategy.

Algorithm 2: Dynamic Encoding

Input: Parameters K, L, N_r, n, all data points in the projected spaces P, sample size n_s
Output: A set of encoded points EP

```
Initialize EP with size n · L · K;
B ← call Breakpoints Selection(K,L,N_r,n,P,n_s);
for i = 1 to L do
    for j = 1 to K do
        for z = 1 to n do
            Use Binarysearch to find b ∈ [0, N_r] such that
              B_ij(b) ≤ h_ij(o_z) ≤ B_ij(b + 1);
            EP_ij(o_z) ← b-th symbol in the 8-bit alphabet;
return EP;
```

Our Method: DET-LSH

- **Indexing phase**

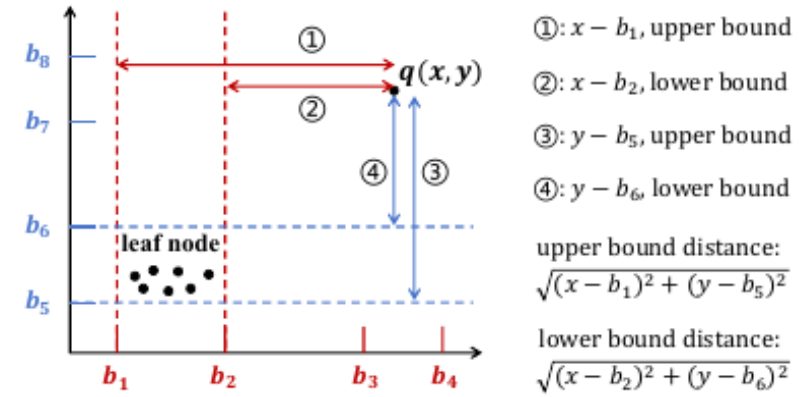
Algorithm 3: Create Index

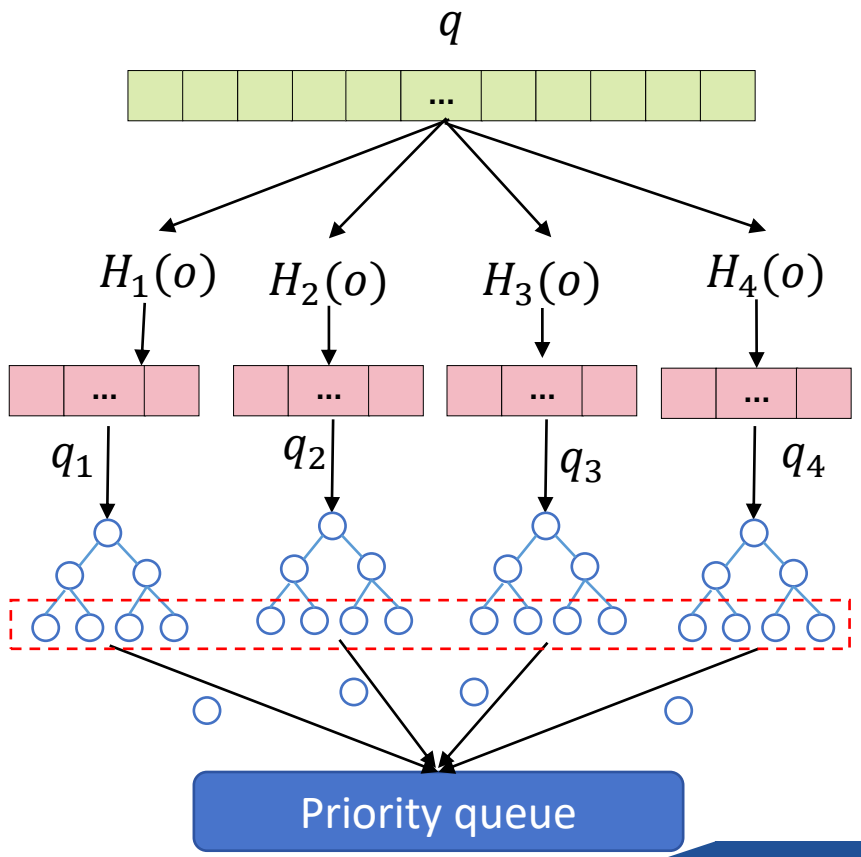
Input: Parameters K, L, n, encoded points set EP, maximum size of a leaf node max_size
Output: A set of DE-Trees: $DETs = [T_1, ..., T_L]$

for $i = 1$ *to* L **do**
  Initialize T_i and generate 2^K first layer nodes as the original leaf nodes;
  for $z = 1$ *to* n **do**
    $ep_i(o_z) \leftarrow (EP_{i1}(o_z), ..., EP_{iK}(o_z))$;
    $pos_z \leftarrow$ the position of o_z in the dataset;
    $target\_leaf \leftarrow$ leaf node of T_i to insert $\langle ep_i(o_z), pos_z \rangle$;
    while $sizeof(target\_leaf) > max\_size$ **do**
      SplitNode($target\_leaf$);
      $target\_leaf \leftarrow$ the new leaf node to insert $\langle ep_i(o_z), pos_z \rangle$;
    Insert $\langle ep_i(o_z), pos_z \rangle$ to $target\_leaf$;
return $DETs$;

Our Method: DET-LSH

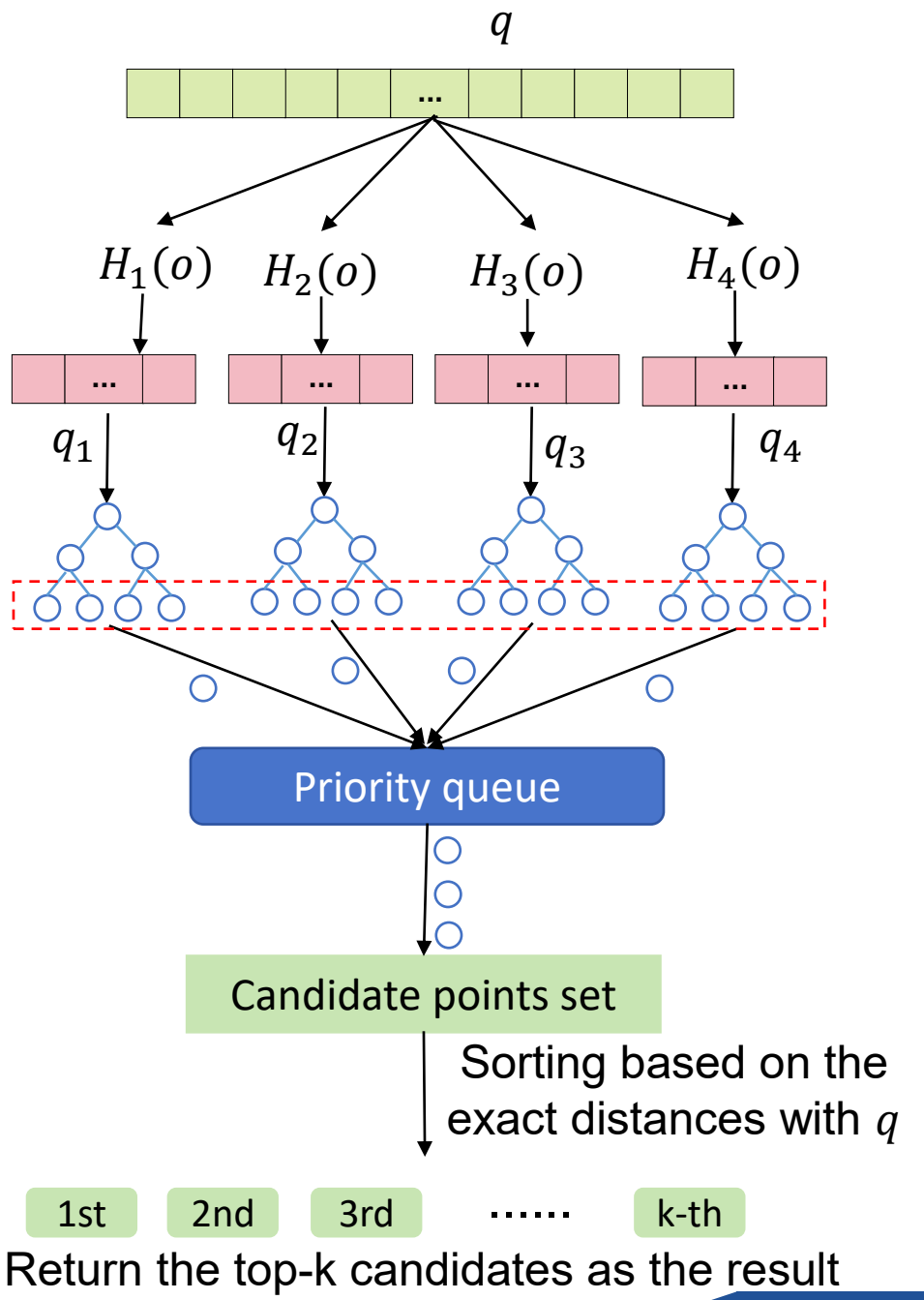
- **Query phase**
  - Input: a search radius r, the maximum number of candidate points βn, a query point q
  - **1. Calculate q_i in all projected spaces (L=4).**
  - **2. Range query:**
    - For each projected space, use q_i to search leaf nodes whose **lower bound distances** with q_i are less than the search radius r
    - Add all obtained leaf nodes to a **priority queue**, and sort them in ascending order according to their lower bound distance with q_i





Our Method: DET-LSH

- **Query phase**
  - **3. Construct candidate points set S:**
    - Continuously pop the leaf node from the priority queue, if its **upper bound distance** with q_i is smaller than r, add all points belong to it into S ; otherwise, traverse data points in the leaf node, and add the data points to S whose distances with q_i are smaller than r.
    - The **termination condition** is that the number of points in S equals to βn (we set $\beta = 0.1$ in practice).
  - **4. Sort and return**
    - Sort and obtain the top-k candidate points in S whose **exact distances with q in the original space** are smaller than other candidate points.
    - Return the top-k candidates as the results.



Theoretical Guarantee

Let $\mathcal{H}_i(o) = [h_{i1}(o), ..., h_{iK}(o)]$ denote a data point o in the i-th projected space, where $i = 1, ..., L$. We define three events as follows:

- **E1:** If there exists a point o satisfying $\|o, q\| \leq r$, then its projected distance to q, i.e., $\|\mathcal{H}_i(o), \mathcal{H}_i(q)\|$, is smaller than ϵr for some $i = 1, ..., L$;
- **E2:** If there exists a point o satisfying $\|o, q\| > cr$, then its projected distance to q, i.e., $\|\mathcal{H}_i(o), \mathcal{H}_i(q)\|$, is smaller than ϵr for some $i = 1, ..., L$;
- **E3:** Fewer than βn points satisfying **E2** in dataset $\mathcal{D}$.

Lemma 3. *Given K and c, setting $L = -\frac{1}{\ln \alpha_1}$ and $\beta = 2 - 2\alpha_2^{-\frac{1}{\ln \alpha_1}}$ such that α_1, α_2, and ϵ satisfy Equation 3, the probability that **E1** occurs is at least $1 - \frac{1}{e}$ and the probability that **E3** occurs is at least $\frac{1}{2}$.*

$$\epsilon^2 = \chi^2_{\alpha_1}(K) = c^2 \cdot \chi^2_{\alpha_2}(K). \tag{3}$$

Proof. Given a point o satisfying $\|o, q\| \leq r$, let $s = \|o, q\|$ and $s'_i = \|\mathcal{H}_i(o), \mathcal{H}_i(q)\|$ denote the distances between o and q in the original space and in the i-th projected space, where $i = 1, ..., L$. From Equation 3, we have $\sqrt{\chi^2_{\alpha_1}(K)} = \epsilon$. For each independent projected space, from Lemma 2, we have $\Pr[s'_i > s\sqrt{\chi^2_{\alpha_1}(K)}] = \Pr[s'_i > \epsilon s] = \alpha_1$. Since $s \leq r$, $\Pr[s'_i > \epsilon r] \leq \alpha_1$. Considering L projected spaces, we have $\Pr[\mathbf{E1}] \geq 1 - \alpha_1^L = 1 - \frac{1}{e}$. Likewise, given a point o satisfying $\|o, q\| > cr$, let $s = \|o, q\|$ and $s'_i = \|\mathcal{H}_i(o), \mathcal{H}_i(q)\|$ denote the distances between o and q in the original space and in the i-th projected space, where $i = 1, ..., L$. From Equation 3, we have $\sqrt{\chi^2_{\alpha_2}(K)} = \frac{\epsilon}{c}$. For each independent projected space, from Lemma 2, we have $\Pr[s'_i > s\sqrt{\chi^2_{\alpha_2}(K)}] = \Pr[s'_i > \frac{\epsilon s}{c}] = \alpha_2$. Since $s > cr$, i.e., $\frac{s}{c} > r$, $\Pr[s'_i > \epsilon r] > \alpha_2$. Considering L projected spaces, we have $\Pr[\mathbf{E2}] \leq 1 - \alpha_2^L$, thus the expected number of such points in dataset $\mathcal{D}$ is upper bounded by $(1 - \alpha_2^L) \cdot n$. By *Markov's inequality*, we have $\Pr[\mathbf{E3}] > 1 - \frac{(1-\alpha_2^L)\cdot n}{\beta n} = \frac{1}{2}$. □

Theorem 2. *Algorithm 7 returns a c^2-k-ANN with at least a constant probability of $\frac{1}{2} - \frac{1}{e}$.*

Proof. We show that when **E1** and **E3** hold at the same time, Algorithm 7 returns a correct c^2-k-ANN result. Let o_i^* be the i-th exact nearest point to q in $\mathcal{D}$, we assume that $r_i^* = \|o_i^*, q\| > r_{min}$, where r_{min} is the initial search radius and $i = 1, ..., k$. We denote the number of points in the candidate set under search radius r as $|S_r|$. Obviously, when enlarging the search radius $r = r_{min}, r_{min} \cdot c, r_{min} \cdot c^2, ...$, there must exist a radius r_0 satisfying $|S_{r_0}| < \beta n + k$ and $|S_{c \cdot r_0}| \geq \beta n + k$. The distribution of r_i^* has three cases:

(1) **Case 1:** If for all $i = 1, ..., k$ satisfying $r_i^* \leq r_0$, which indicates the range queries in all L index trees have been executed at $r = r_0$ (line 3-8). Due to **E1**, all r_i^* must in S_{r_0}. Since $S_{r_0} \subsetneq S_{c \cdot r_0}$, all r_i^* also must in $S_{c \cdot r_0}$. Therefore, Algorithm 7 returns the exact k nearest points o_i^* to q.

(2) **Case 2:** If for all $i = 1, ..., k$ satisfying $r_i^* > r_0$, all r_i^* not belong to S_{r_0}. Since Algorithm 7 may terminate after executing range queries in part of L index trees at $r = c \cdot r_0$ (line 8), we cannot guarantee that $r_i^* \leq c \cdot r_0$. However, due to **E3**, there are at least k points o_i in $S_{c \cdot r_0}$ satisfying $\|o_i, q\| \leq c^2 r_0$, $i = 1, ..., k$. Therefore, we have $\|o_i, q\| \leq c^2 r_0 \leq c^2 r_i^*$, i.e., each o_i is a c^2-ANN point for corresponding o_i^*.

(3) **Case 3:** If there exists an integer $m \in (1, k)$ such that for all $i = 1, ..., m$ satisfying $r_i^* \leq r_0$ and for all $i = m + 1, ..., k$ satisfying $r_i^* > r_0$, indicating that **Case 3** is a combination of **Case 1** and **Case 2**. For each $i \in [1, m]$, Algorithm 7 returns the exact nearest point o_i^* to q based on **Case 1**. For each $i \in [m+1, k]$, Algorithm 7 returns a c^2-ANN point for o_i^* based on **Case 2**.

Therefore, when **E1** and **E3** hold simultaneously, Algorithm 7 can always correctly answer a c^2-k-ANN query, i.e., Algorithm 7 returns a c^2-k-ANN with at least a constant probability of $\frac{1}{2} - \frac{1}{e}$. □

Evaluation: DET-LSH v.s. LSH-based methods

Table 2: Datasets

Dataset	Cardinality	Dimensions	Type
Msong	994,185	420	Audio
Deep1M	1,000,000	256	Image
Sift10M	10,000,000	128	Image
TinyImages80M	79.302,017	384	Image
Sift100M	100,000,000	128	Image
Yandex Deep500M	500,000,000	96	Image
Microsoft SPACEV500M	500,000,000	100	Text
Microsoft Turing-ANNS500M	500,000,000	100	Text

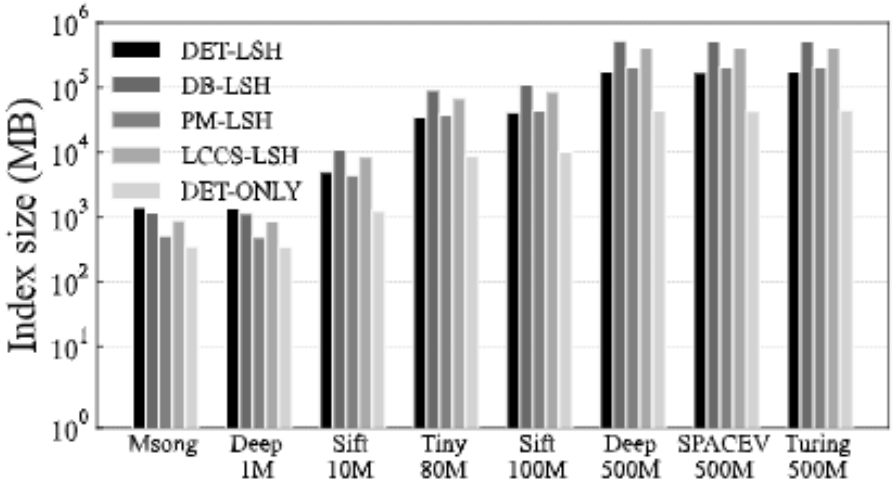


Figure 6: Index size for all datasets.

Table 3: Performance comparison with competitors (the best value in each row is highlighted in bold; the number in parentheses indicates how many times slower a method is than the best method).

		DET-LSH	DB-LSH	PM-LSH	LCCS-LSH	DET-ONLY
Msong	Query Time (ms)	112.97 (1.43)	118.10 (1.49)	120.36 (1.53)	170.13 (2.16)	**78.87**
	Recall	**0.9546**	0.9474	0.949	0.849	0.891
	Overall Ratio	**1.0012**	1.0013	1.0013	1.0035	1.0046
	Indexing Time (s)	4.654 (3.90)	4.974 (4.17)	2.950 (2.47)	28.925 (24.2)	**1.194**
Deep1M	Query Time (ms)	109.28 (1.37)	117.79 (1.48)	207.37 (2.61)	136.21 (1.71)	**79.51**
	Recall	**0.9112**	0.8552	0.857	0.848	0.818
	Overall Ratio	**1.0022**	1.0038	1.0042	1.0039	1.0061
	Indexing Time (s)	4.647 (3.97)	4.809 (4.10)	2.991 (2.55)	57.652 (49.2)	**1.172**
Sift10M	Query Time (ms)	506.34 (1.20)	944.23 (2.25)	1482.84 (3.53)	1905.09 (4.53)	**420.43**
	Recall	**0.9644**	0.9438	0.9338	0.8924	0.886
	Overall Ratio	**1.0009**	1.0015	1.0016	1.0021	1.0035
	Indexing Time (s)	44.435 (4.00)	64.861 (5.85)	80.099 (7.22)	509.417 (45.9)	**11.094**
TinyImages80M	Query Time (ms)	7676.23 (1.00)	8164.96 (1.07)	13672.6 (1.79)	11272.8 (1.47)	**7657.08**
	Recall	**0.9108**	0.9056	0.8822	0.87	0.8338
	Overall Ratio	**1.0016**	**1.0016**	1.0023	1.0019	1.0036
	Indexing Time (s)	335.419 (4.08)	641.988 (7.81)	1471.31 (17.89)	12128.1 (147.5)	**82.235**
Sift100M	Query Time (ms)	4757.76 (1.20)	11064.8 (2.78)	15722.8 (3.95)	24221.8 (6.08)	**3983.41**
	Recall	**0.9822**	0.9652	0.944	0.892	0.8848
	Overall Ratio	**1.0005**	1.0007	1.0013	1.0019	1.0034
	Indexing Time (s)	439.434 (4.04)	952.773 (8.76)	1922.7 (17.67)	7519.43 (69.1)	**108.782**
Yandex Deep500M	Query Time (ms)	28546.6 (1.09)	61657.9 (2.35)	91724.2 (3.50)	62411.8 (2.38)	**26200.4**
	Recall	**0.9852**	0.9644	0.9298	0.9506	0.9176
	Overall Ratio	**1.0003**	1.0009	1.0032	1.0009	1.0058
	Indexing Time (s)	2263.87 (4.22)	17182.7 (32.04)	13685.2 (25.52)	85968.3 (160.3)	**536.262**
Microsoft SPACEV500M	Query Time (ms)	31404.3 (1.07)	66632.3 (2.28)	94868.3 (3.25)	70697.5 (2.42)	**29212.6**
	Recall	**0.963**	0.9492	0.9568	0.9198	0.8978
	Overall Ratio	**1.0008**	1.0012	1.0011	1.0026	1.00336
	Indexing Time (s)	2204.94 (4.21)	16114.7 (30.77)	13189.5 (25.19)	87591.1 (167.3)	**523.662**
Microsoft Turing-ANNS500M	Query Time (ms)	31280.1 (1.04)	68636.6 (2.28)	106987 (3.55)	73618.2 (2.44)	**30127.2**
	Recall	**0.9806**	0.9604	0.9636	0.9404	0.9008
	Overall Ratio	**1.0005**	1.0012	1.0009	1.0012	1.0043
	Indexing Time (s)	2301.02 (4.22)	16408.2 (30.11)	12680.2 (23.27)	79162.5 (145.3)	**545.006**

While achieving better query accuracy than competitors, DET-LSH achieves up to **6x speedup in indexing time and 2x speedup in query time** over the state-of-the-art LSH-based methods.

Evaluation: DET-LSH v.s. LSH-based methods

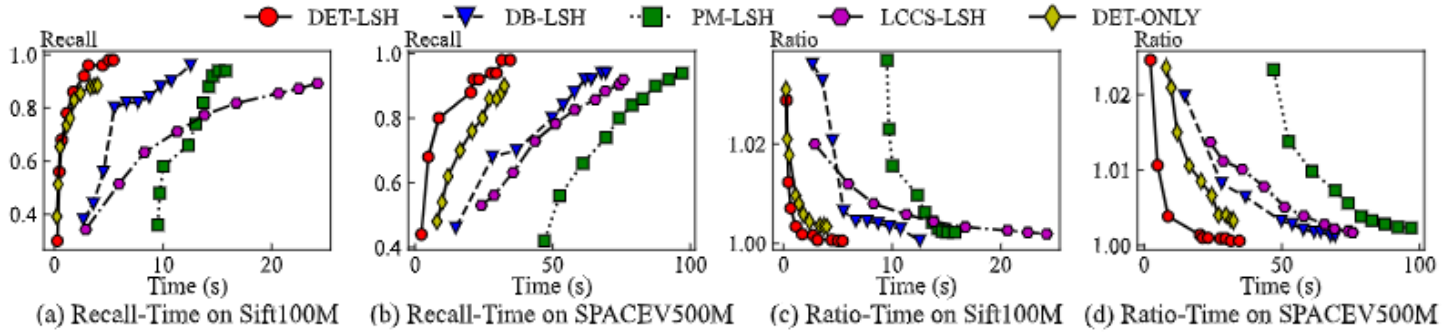


Figure 7: Recall-time and overall ratio-time curves.

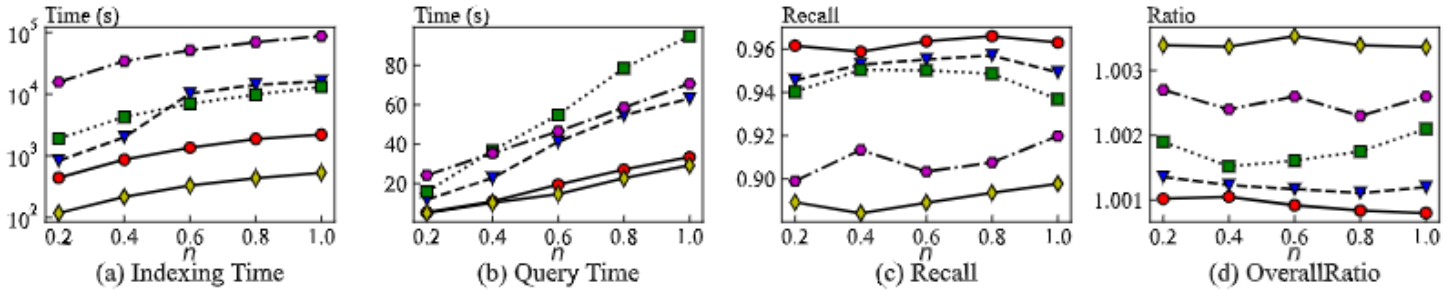


Figure 8: Scalability: performance under different n on Microsoft SPACEV500M.

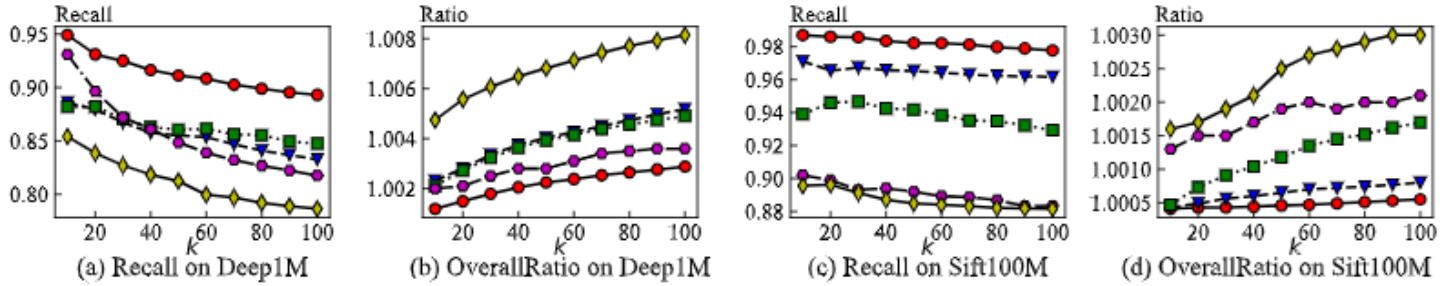


Figure 9: Performance under different k.

Evaluation: DET-LSH v.s. Graph-based methods

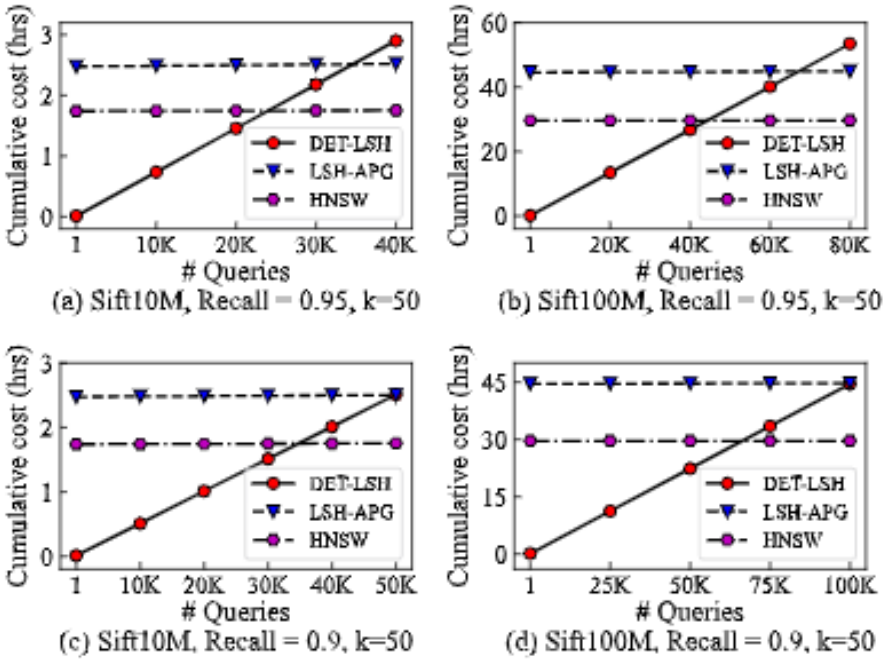


Figure 10: Cumulative query cost (first query includes indexing time).

DET-LSH has an advantage in indexing efficiency: it creates the index and answers 30K-70K queries before the best competitor (i.e., HNSW) answers its first query.

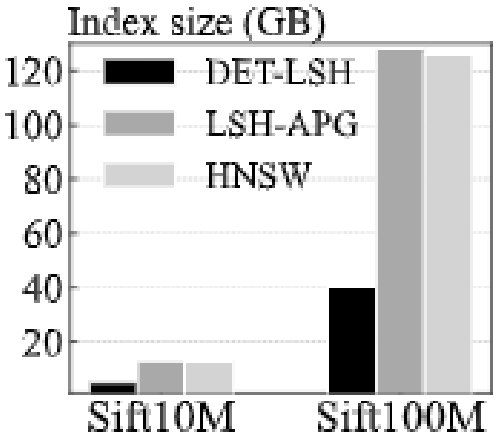


Figure 11: Index size.

DET-LSH's index is almost **3x smaller** in size than the index constructed by the competitors.

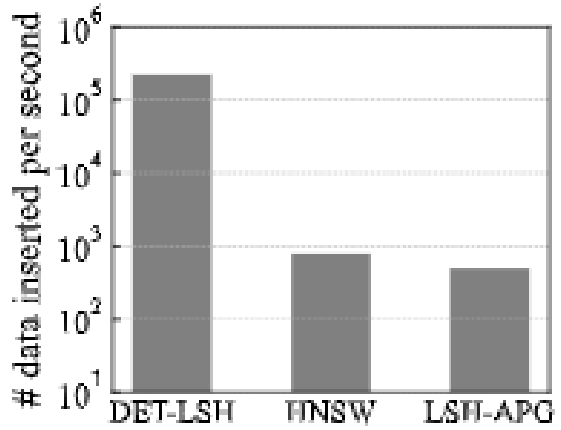


Figure 12: Update efficiency.

In this scenario that involves updates, DET-LSH is **2-3 orders of magnitude faster** than HNSW and LSH-APG.

中国科学院计算技术研究所

INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES