

LIRA: A Learning-based Query-aware Partition Framework for Large-scale ANN Search

Ximu Zeng
University of Electronic Science and
Technology of China
Chengdu, China
ximuzeng@std.uestc.edu.cn

Liwei Deng
University of Electronic Science and
Technology of China
Chengdu, China
deng_liwei@std.uestc.edu.cn

Penghao Chen
University of Electronic Science and
Technology of China
Chengdu, China
penghaochen@std.uestc.edu.cn

Xu Chen
University of Electronic Science and
Technology of China
Chengdu, China
xuchen@std.uestc.edu.cn

Han Su*
Yangtze Delta Region Institute
(Quzhou), University of Electronic
Science and Technology of China
Quzhou, China
hansu@uestc.edu.cn

Kai Zheng*[†]
University of Electronic Science and
Technology of China
Chengdu, China
zhengkai@uestc.edu.cn

Abstract

Approximate nearest neighbor search is fundamental in information retrieval. Previous partition-based methods enhance search efficiency by probing partial partitions, yet they face two common issues. In the query phase, a common strategy is to probe partitions based on the distance ranks of a query to partition centroids, which inevitably probes irrelevant partitions as it ignores data distribution. In the partition construction phase, all partition-based methods face the boundary problem that separates a query's nearest neighbors to multiple partitions, resulting in a long-tailed k NN distribution and degrading the optimal $nprobe$ (i.e., the number of probing partitions). To address this gap, we propose *LIRA*, a Learning-based Qery-aware partition framework. Specifically, we propose a probing model to directly probe the partitions containing the k NN of a query, which can reduce probing waste and allow for query-aware probing with $nprobe$ individually. Moreover, we incorporate the probing model into a learning-based redundancy strategy to mitigate the adverse impact of the long-tailed k NN distribution on search efficiency. Extensive experiments on real-world vector datasets demonstrate the superiority of *LIRA* in the trade-off among accuracy, latency, and query fan-out. The codes are available at <https://github.com/SimoneZeng/LIRA-ANN-search>.

CCS Concepts

• **Information systems** → **Information retrieval query processing**; **Learning to rank**; *Top-k retrieval in databases*.

*Corresponding authors: Han Su and Kai Zheng.

[†]Kai Zheng is with Yangtze Delta Region Institute (Quzhou), School of Computer Science and Engineering, UESTC. He is also with Shenzhen Institute for Advanced Study, UESTC.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WWW '25, April 28-May 2, 2025, Sydney, NSW, Australia

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1274-6/25/04

<https://doi.org/10.1145/3696410.3714633>

Keywords

Approximate nearest neighbor search, Learning-to-index

ACM Reference Format:

Ximu Zeng, Liwei Deng, Penghao Chen, Xu Chen, Han Su, and Kai Zheng. 2025. LIRA: A Learning-based Query-aware Partition Framework for Large-scale ANN Search. In *Proceedings of the ACM Web Conference 2025 (WWW '25)*, April 28-May 2, 2025, Sydney, NSW, Australia. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3696410.3714633>

1 Introduction

The nearest neighbor (NN) search is well studied in the community of information retrieval [38, 46, 53, 58]. By embedding unstructured data (e.g., texts and images) into vectors [30, 41], the similarity of vectors represents semantic similarity [49, 51]. Hence, vector space search is fundamental for efficiently retrieving large-scale unstructured data in retrieval-augmented generation [6, 17, 27, 44, 45]. Given a vector dataset \mathcal{D} and a query vector q , the goal of k nearest neighbors (k NN) search is to find the k vectors nearest to q from the dataset. However, the *exact* NN search is time-consuming with the growth of dataset cardinalities and dimensions [2, 25]. Consequently, current works shift focus towards approximate nearest neighbor (ANN) search [5, 8, 34], which seeks a trade-off between latency and accuracy by retrieving with indexing techniques.

1.1 Prior Approaches and Limitations

Partition-based methods are the backbone of ANN search [6, 17, 42], which are suitable for partial data loading [16, 22]. The k NN of a query q can be separated into several partitions. We denote the partitions containing the k NN of query q as its k NN **partitions**, which means these partitions should be probed to retrieve the exact top- k nearest neighbors. A lower $nprobe$ (i.e., the number of probing partitions, also known as the query fan-out) is preferred for scalability [39, 55]. A naive way to achieve a low $nprobe$ can be partition pruning, but a trade-off exists between fewer probing partitions and high recall. If the probing partitions poorly cover the k NN partitions, the recall of retrieval results drops. If probing partitions without k NN, search efficiency degenerates. We call this phenomenon the **Curse of Partition Pruning**.

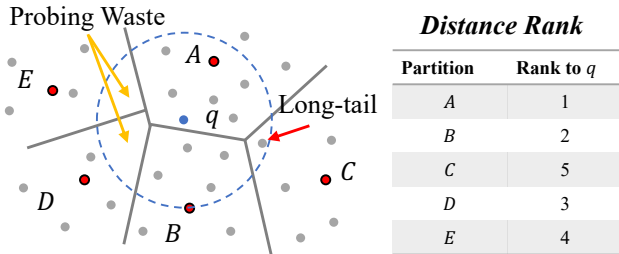


Figure 1: Example for probing waste. The blue point is a query, and the red points are the centroids of partitions.

Here, we go through the limitations of previous studies in partition pruning. Considering the low accuracy of tree-based [12] and hash-based [40, 43, 46, 47] methods, clustering methods are promising in dealing with the dilemma of partition pruning. The inverted file (IVF) index [25] builds clusters with the K-Means algorithm and then searches in fixed $nprobe$ nearest partitions **according to the distance rank between a query and the cluster centroids**. Further, BLISS [18] build partitions with deep learning models but still set a fixed $nprobe$ for all queries to achieve the trade-off between search latency and recall. However, since no fixed $nprobe$ can fit all queries [57], there remain limitations in search performance.

Limit 1. Partition pruning with the distance rank to partition centroids wastes $nprobe$. As shown in a toy example in Fig. 1, suppose the data points of a dataset are divided into partitions. The top 10 NNs of a query q are distributed in three partitions (i.e., partition A, B, and C), where partition C ranked as the fifth nearest partition of q . To ensure all 10 NNs are included, the minimum number of probing partitions based on the centroid distance rank is $nprobe = 5$. Alternatively, a more cost-effective way is to directly probe the three kNN partitions, resulting in the optimal $nprobe$ to 3. Hence, pruning partitions according to the centroid distance rank still wastes $nprobe$, and such probing waste is ubiquitous in high-dimension datasets as illustrated in Section 2.

Limit 2. Hard partitioning cannot inherently achieve low $nprobe$ due to the long-tailed distribution of kNN . Partitioning strategies aim to reduce $nprobe$. However, due to the curse of dimension and local density variations, the kNN distribution with hard partitioning methods (i.e., each data point is put in one partition) often exhibits the notorious long-tailed characteristic. Specifically, while most kNN may be densely located in a few partitions, the remaining kNN scatter across many other partitions [21]. As the example shown in Fig. 1, the top-10 kNN of a query q are distributed as [5, 4, 1, 0, 0] in five partitions. The scattered one kNN can be regarded as a long-tail kNN , resulting in a less efficient search process. Due to this limitation, BLISS [18] constructs four groups of independent partitions and indexes. Hence, the long-tailed distribution of kNN diminishes the cost-effectiveness of probing partitions and increases the query fan-out undesirably.

1.2 Our Solution

We find that the essential issue of the limitations mentioned above stems from the distribution of kNN in partitions. Hence, we improve the performance of partition-based ANN search from two aspects, query process and index construction.

Insight 1. A meta index that directly probes the kNN partitions is required. To avoid confusion, we define the index for

inter-partitions that facilitates partition pruning as the **meta index**, and the index for intra-partition (i.e., the index that only organizes the data points within one partition) as **internal index**. In this study, we focus on the optimization of meta index among the two-level index for partitions, in which the internal index can apply any existing index structure such as HNSW [35]. As mentioned in Limit 1, the optimal number of probing partitions for a query q , denoted as $(nprobe^q)^*$, is exactly the number of its kNN partitions. When the context is clear, we refer to $(nprobe^q)^*$ as $nprobe^*$ in this paper for brevity. To address the probing waste through an effective query process, the ideal meta index needs to directly probe the kNN partitions of a query. Compared to IVF, the meta index can achieve high recall while reducing the $nprobe$ simultaneously. **Insight 2. Redundant partitioning is required to mitigate the long-tailed kNN distribution and further reduce $nprobe^*$.** As discussed in Limit 2, the search inefficiency often arises from the long-tailed kNN distribution. To address this from the aspects of index construction, a feasible approach is to redundantly put a query’s long-tail kNN into other densely distributed kNN partitions. For instance, the initial optimal $nprobe$ of q in Fig. 1 is $nprobe^* = 3$. By introducing redundancy - duplicating the single kNN in partition C to another kNN partition (i.e., partition A or B) - the revised optimal $nprobe$ of q can be further reduced to $nprobe^* = 2$, since merely probing partitions A and B suffices to cover all the top-10 kNN . However, instead of merely reducing the $nprobe$, the objective in partition-based ANN search is striking a better trade-off between latency and recall. The search latency can potentially increase with more data replicas. Hence, we need an exquisite redundant partition method to balance partition pruning and data redundancy.

Based on the above insights, we propose *LIRA*, a Learning-based queRy-aware partition framework to serve as a meta index across partitions. In general, *LIRA* follows the “one size does not fit all” principle and explores the power of adaptation. After building initial clustered partitions, we utilize a learning model to infer where to probe for individual queries, which data points need to be duplicated, and where to duplicate these data points across partitions. Specifically, we first enhance the query process by developing a probing model to predict query-dependent kNN partitions, thereby enabling precise partition pruning with less probing waste. The probing model uses a tunable threshold on its output probabilities, allowing for more adaptive partition pruning and fine-grained tuning than the $nprobe$ configuration in IVF. Second, we improve the construction of partitions by efficiently duplicating data points with the probing model. When building redundant partitions, we novelly transfer the task from an exhaustive search for kNN of all data points globally to discriminating data points individually. Finally, in the top-k retrieval phase, *LIRA* leverages the model’s probing probabilities to guide searching across partitions and to reduce query fan-out. In summary, we make the following contributions.

- To save the probing waste in a query-aware way, we propose a learning-based partition pruning strategy where a probing model generates the probing probabilities of partitions for each query.
- To mitigate the effect of long-tailed kNN distribution by building redundant partitions, we novelly transfer the problem of duplicating data points globally to individually and then propose a learning-based redundancy strategy with the probing model.

- We conduct extensive experiments on publicly available high-dimensional vector datasets, demonstrating the superiority of *LIRA* in recall, latency, and partition pruning.

2 Preliminaries

2.1 Definitions

Suppose $\mathcal{D} = \{v^1, v^2, \dots, v^N\}$ be a dataset of N d -dimension data points separated in B partitions, and $dist(v^1, v^2)$ is the function to calculate the distance between data points v^1 and v^2 . We define the ANN search problem in the partition-based scenario as follows.

DEFINITION 1 (kNN COUNT DISTRIBUTION). Given a query vector q , let $S_{GT} = \{v^1, v^2, \dots, v^k\}$ be the ground truth (abbreviated as GT) set of q 's k nearest neighbors. Let n_i^q be the count of ground truth k NN of q in the i -th partition, the k NN count distribution of a query q can be defined as $n^q = [n_1^q, n_2^q, \dots, n_B^q]$, where $\sum_{i=1}^B n_i^q = k$.

DEFINITION 2 (RECALL@K). Recall@ k refers to the proportion of the ground truth top- k nearest neighbors retrieved by an ANN search method out of all ground truth k NN in the dataset.

$$Recall@k = \frac{|S \cap S_{GT}|}{k} \times 100\%. \quad (1)$$

where S is the retrieved results. A higher Recall@ k value indicates a greater number of exact top- k nearest neighbors are retrieved.

DEFINITION 3 (LONG-TAIL DATA POINT). Given a k NN count distribution of a query q , $n^q = [n_1^q, n_2^q, \dots, n_B^q]$, we regard the part of k NN with $n_i^q = 1$ as the long-tail part in the k NN count distribution. The specific data point served as the long-tail k NN of q where $n_i^q = 1$ in the long-tail part is termed as a long-tail data point.

For a k NN count distribution n^q , we term the partition that contains the ground truth k NN as a **k NN partition**. We denote the k NN partition distribution $p^q = [p_1^q, p_2^q, \dots, p_B^q]$ as a binary mask over k NN count distribution, where k NN partitions are marked with 1 while others with 0, and $\sum_{i=1}^B p_i^q = (nprobe^q)^*$. In addition, for a long-tailed k NN count distribution n^q , we regard partitions with $n_i^q > 1$ as the **replica partition** (e.g., partition A and B in Fig. 1). According to Insight 2, duplicating a long-tail data point into its replica partitions can save one probing and reduce $nprobe^*$ further. There may be other replica partitions of a data point, since it can be in the long-tail part of the k NN distribution of other queries.

DEFINITION 4 (OBJECTIVE). Compared with baseline methods under equivalent Recall@ k , our objective is to optimize the partition pruning and query latency. Our approach involves the refinement of partition by integrating a learning-based redundancy strategy and query-aware retrieval process with a learned probing model.

2.2 Motivation

We provide motivations through conducting preliminary studies on the SIFT [2] 1M dataset with 10K queries. We illustrate the waste of probing caused by partition pruning with centroids distance ranks, and common long-tail k NN in k NN count distributions.

Probing Waste with Distance Ranking. In IVF, the probing cardinality $nprobe$ infers probing the nearest $nprobe$ partitions. Ideally, the optimal number of probing partitions, $nprobe^*$, should be no larger than k for Recall@ k . We define $nprobe_{dist}^*$ as the maximum

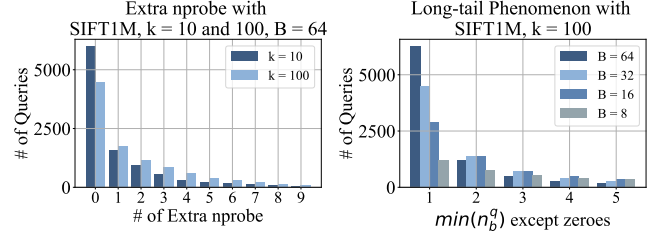


Figure 2: Extra probing with distance ranking (LEFT) and common phenomenon of long-tail k NN (RIGHT).

distance rank among the k NN partitions, which implies the nearest $nprobe_{dist}^*$ partitions need to be probed to cover all k NN. However, we find that probing according to $nprobe_{dist}^*$ wastes probing cardinality. For example, the $nprobe_{dist}^*$ exceeds 20 for some queries when retrieving top-10 k NN and the waste of probing is even worse with $k = 100$ (See details in Appendix A.2). In addition, we show the extra $nprobe$ for each query when probing partitions according to the distance rank in Fig. 2 (LEFT). The extra $nprobe$ is the difference between the optimal $(nprobe^q)^*$ and the $(nprobe^q)_{dist}^*$. Hence, these observations suggest an opportunity to reduce the probing waste by refining the probing strategy in the query phase. **Common Long-tail k NN.** To discover the ubiquity of long-tailed k NN count distribution, we analyze the long-tail phenomenon by calculating the minimum n_i^q in a query q 's k NN count distribution expect zeros. Considering the k NN of a query can be more congested with larger partition sizes, we set the number of partitions $B \in \{64, 32, 16, 8\}$, respectively. As depicted in Fig. 2 (RIGHT), the long-tail phenomenon exists regardless of the number of partitions. In detail, as we can see in the horizontal axis with a value of 1, thousands of queries have long-tailed k NN count distribution among a total 10k queries. Hence, if valuable knowledge can be extracted from the k NN count distributions, there are two opportunities to improve the ANN search by incorporating redundancy in partition building. First, we can infer the latent long-tail data points in the dataset. Second, we can predict the replica partitions for long-tail data points. In Section. 3.3, we present the time complexity of building redundant partitions with global k NN count distributions of all data, and introduce an efficient learning-based redundancy strategy.

3 Method

In this section, we first present an overview of *LIRA*. We then introduce the partition pruning strategy with a learned probing model. We illustrate the learning-based redundancy strategy by identifying long-tail data points and duplicating them to replica partitions with the probing model. Finally, we present k NN retrieval across partitions by using the probing model as the meta index.

3.1 Framework Overview

The workflow of *LIRA* can be divided into two distinct processes: the construction of redundant partitions and the top- k retrieval for queries, which we illustrate through a toy example with 5 partitions in Fig. 3 and 6, respectively. **(1) Probing model training.** After initializing the B partitions with the vanilla K-Means algorithm,

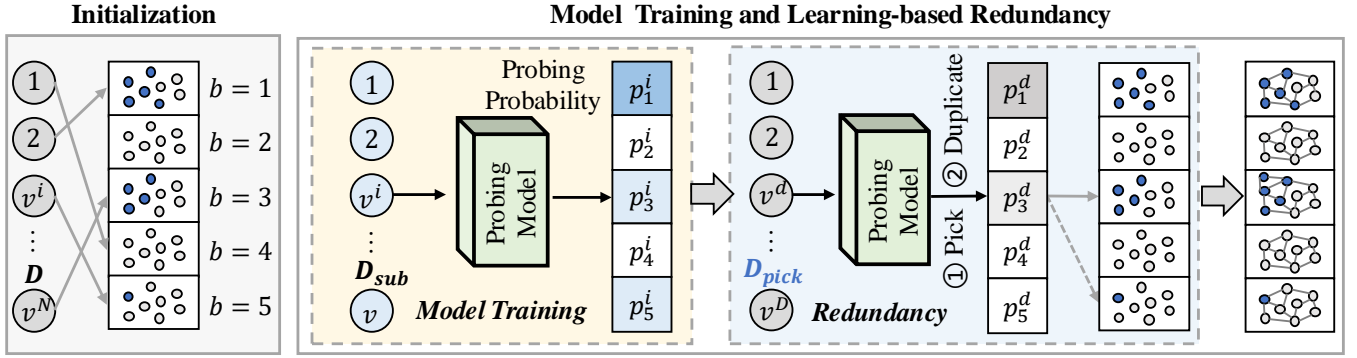


Figure 3: Partition initialization (LEFT), the probing model training (MIDDLE) and learning-based redundancy strategy (RIGHT).

LIRA targets training a probing model to serve as the meta index and to engage in building redundant partitions. Specifically, the probing model is applied threefold in *LIRA*: learning the mapping function $f(\cdot)$ of data points to k NN partitions in training, providing potential long-tail data points and replica partitions in learning-based redundancy, and guiding the partition probing during the query-aware retrieval process. (2) **Learning-based redundancy.** During the redundancy phase, we aim to duplicate the potential long-tailed data points to the replica partitions by using the probing model. Finding the deep correlation between the k NN partitions and the replica partitions, we novelly transform the problem of picking and duplicating long-tailed data points from globally to individually. First, we pick the data points with more predicted $nprobe$ as the potential long-tail data points. Second, we choose the partition with a high predicted possibility in \widehat{p}^v to put the replica of data point v . We construct the internal indexes for each partition individually after the redundant partitions are built (See detailed algorithm of the two-level index in Appendix A.3). (3) **Query-aware retrieval process.** Since the probing model can map a data point in the vector space to its k NN partitions, we can use it to guide the top- k retrieval across partitions. The predicted probabilities are utilized along with a probability threshold in the inter-partition pruning, and the internal indexes are used for inner-partition searching.

3.2 Probing Model Training

Probing Model. The model has two inputs: (1) the query vector q , and (2) the distances between q and partitions centroids I . We can regard the probing model as a multivariate binary classifier for whether to probe each partition. The output is the predicted probability \widehat{p} in the dimension of the number of partitions B . Hence, the model can be represented as $f(q, I) = \widehat{p}$. We convert the two inputs as feature vectors x_q and x_I with individual networks, respectively, and then concatenate the two feature vectors to generate the predicted probing probabilities \widehat{p} as follows.

$$x_q = \phi_q(q), x_I = \phi_I(I), \widehat{p} = \phi_p(x_q \oplus x_I) \quad (2)$$

where ϕ_q, ϕ_I, ϕ_p are three independent multi-layer models, and the output of ϕ_p is the predicted probabilities for probing partition \widehat{p} .

As discussed in Section 2.2, an ideal probing model served as the meta index should directly probe k NN partitions of a query regardless of its distance rankings to cluster centers. Hence, the

labels of a q are the same as the k NN partition distribution p^q , where partitions with $n_i^q > 0$ are regarded as positive and other partitions with no k NN are labeled as negative. For example, the labels of a k NN count distribution $[5, 4, 1, 0, 0]$ is $[1, 1, 1, 0, 0]$.

Network Training. We sample a subset of data from the whole dataset D as training data and use the provided queries of a dataset to evaluate the effectiveness of *LIRA* (See detailed description on scalability in Appendix A.3 and A.5). For each training data, we search the k NN from the training data to get the k NN partition distribution p^q . The output of the probing model \widehat{p}_b^q in $[0, 1]$ is the possibility of probing each partition. We take the partition with $\widehat{p}_b^q \geq \sigma$ as a probing partition to support query-aware $nprobe$. The σ is set as 0.5 in training and is tunable in the query process, which provides fine-grained tuning in partition pruning than the $nprobe$ configuration in IVF. Hence, we can solve the multivariate binary classification problem with the cross-entropy loss:

$$\mathcal{L}(p^q, \widehat{p}^q) = - \sum_{b=1}^B (p_b^q \cdot \log(\widehat{p}_b^q) + (1 - p_b^q) \cdot \log(1 - \widehat{p}_b^q)) \quad (3)$$

where the $\mathcal{L}(p^q, \widehat{p}^q)$ is the loss of the probing model on a query q .

3.3 Learning-based Redundancy

Redundancy is a crucial step in *LIRA* since it helps refine the initial hard partitions as redundant partitions. The aim of redundancy is to reduce the side-effect of long-tailed k NN count distribution by reasonably duplicating each long-tail data point into one of its replica partitions. There are two important issues about redundancy. (1) Under the initial partition, how do we identify data points that tend to be the long-tail data points in the k NN count distribution of queries? (2) To duplicate the long-tail data points, which partitions should we transfer these data points into?

Pick Data Points. A data point v might be a long-tail data point for any query. To identify whether a data point is long-tail, it is necessary to examine the k NN counts distribution, n^q , for all queries. Since the n^q is unknown when building redundant partitions, we can only use the k NN count distribution of the data itself, n^v , which involves finding the data points in the long-tail k NN parts of other data's k NN count distribution. Duplicating globally means computing the k NN of all data to identify whether a data point is long-tail. However, the computation cost of getting k NN of the whole data

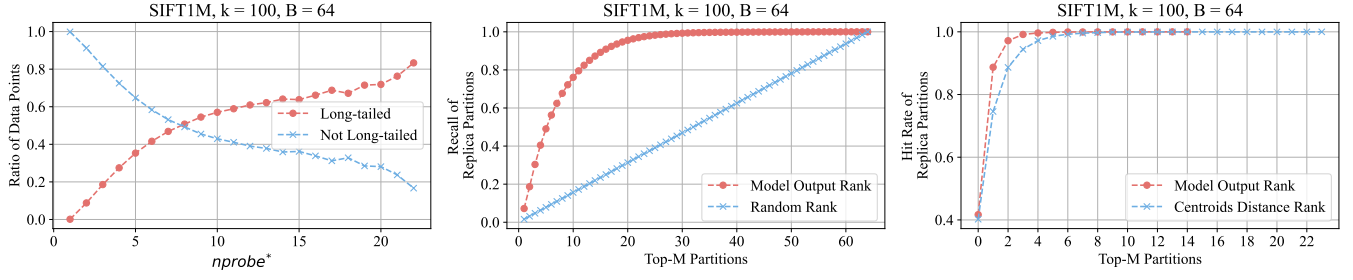


Figure 4: Ratio of long-tailed and not long-tailed data points under certain $nprobe^*$ (LEFT). The recall (MIDDLE) and hit rate (RIGHT) of replica partitions among top-M partitions with model output rank or centroids distance rank.

		① Pick Data	② Duplicate Data				
		Long-tail	Centroids Distance Rank				
Replicate Globally 	v^1	✗	2		3	4	5
	v^2	✓	5	3	2		4
	v^3	✓	2		3	5	4
		$nprobe^*$	Model Output Rank				
Replicate Individually 	v^1	1	3	1	2	4	5
	v^2	4	4	3	5	1	2
	v^3	3	1	4	2	3	5

Replica Partitions kNN Partitions
 Located Partition Partition to Put Replica

Figure 5: Pick and duplicate potential long-tailed data points individually with the probing model is more efficient than using ground truth kNN count distribution globally.

$O(N^2 \cdot d)$ is unacceptable in large-scale datasets. Consequently, it is impractical to identify all the long-tail data points globally. Innovatively, we circumvent this challenge and transfer the issue of identifying long-tail data points from globally to individually.

Specifically, we observe an interesting phenomenon from empirical analysis: **Data points with a larger $nprobe^*$ are more likely to be long-tail data points.** In detail, we record the kNN count distribution and $nprobe^*$ of individual data in SIFT and find the long-tail kNN data points. Varying the specific $nprobe^*$, we calculate the ratio of data that identified as long-tail data points versus those are not. As demonstrated in Fig. 4 (LEFT), an increase in $nprobe^*$ correlates with a higher ratio of long-tail data points. The observation aligns with the spatial partitioning in vector space: data points with kNN separated across multiple partitions are more likely located at the boundaries of partitions and thus are more prone to being long-tail.

In Fig. 5, we illustrate the transformation of picking data points by using other data’s kNN globally to using ego kNN (i.e., the kNN of a data point itself) individually. For example, long-tail data points, v^2 and v^3 , exhibit a higher $nprobe^*$ compared to the non-long-tail data point v^1 . Leveraging the accurate prediction of the probing model, we can reliably use it to estimate the $nprobe^*$ of data points and pick potential long-tail data points individually. We apply the model to get the \hat{p} of all data points, selecting those within the upper

η percentile of predicted $nprobe$. Hence, utilizing the probing model obviates the need to find kNN on whole data globally, streamlining the process of identifying long-tail data points.

Duplicate Data Points. After identifying data points requiring duplication, the next challenge is selecting appropriate partitions to put these replicas. Similar to the challenge of high computation cost in picking data points, the replica partitions of each long-tail data point are unknown if the global kNN count distribution is inaccessible. For all the long-tail data points, we record the kNN partitions and replica partitions, and we observe an interesting phenomenon: **the replica partitions for duplicating a data point v have a strong relationship to its kNN partitions.** In detail, most of the replica partitions of v align closely with its kNN partitions. As the example in duplicating data of Fig. 5, the kNN partition (i.e., the four partitions depicted in blue) of a long-tail data point, v^2 , can cover its replica partitions (i.e., the three partitions shown in yellow). This provides a promising approach to getting replica partitions by leveraging predicted probing partitions from the model.

Another problem follows this insight: since the model can produce many probing partitions for a data point v , which one should be chosen to put the replica of v ? Our analysis indicates that **the partition b with higher probing probability p_b^v is more likely to be a replica partition for v .** In detail, we first get the replica partitions of all the long-tail data points. Then, we calculate $Recall_{rep}^v$, the recall between replica partitions and the top- M predicted partitions, where M ranges from 1 to B .

$$Recall_{rep}^v = \frac{|S_{model}^v \cap S_{rep}^v|}{|S_{rep}^v|} \times 100\%. \quad (4)$$

where S_{model}^v is the set of top- M partitions in the model output of data point v , and S_{rep}^v is v ’s set of the replica partitions. For comparison, we also evaluate a random ranking of partitions represented by the blue line. (1) As shown in Fig. 4 (MIDDLE), we can see that the predicted probing partitions can effectively cover the replica partition, for $Recall_{rep}$ increases to nearly 1 with just $M = 20$. (2) In addition, the gradually decreasing slope of the red line can support that partitions with high output probabilities tend to be replica partitions. Hence, this insight informs our duplication strategy, which utilizes output probing probability. As illustrated in Fig. 5, if a long-tail data point is not in the partition with the highest output rank, we duplicate it into this partition; otherwise, we put it into the partition with the second-highest output rank.

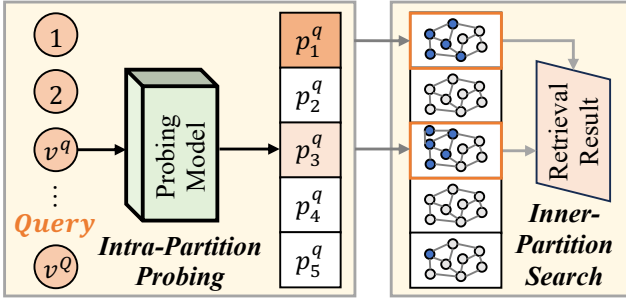


Figure 6: Retrieval process across partitions.

Furthermore, compared with duplicating long-tail data points according to the distance rank of partition centroids, we observe that using the model output rank is more valid. Typically, centroid distance ranks are considered when duplicating data points. For example, a data point can be duplicated up to 8 times in the closure clustering assignment in SPANN [6]. As we highlight in Limit 1, using centroid distance ranks often wastes $nprobe$ for retrieving kNN . This limitation also emerges when choosing partitions to duplicate long-tail data points. For example, the long-tail data point v^3 in Fig. 5 has two replica partitions. The distance rank of replica partitions for v^3 is (2, 3), while the output rank can be (1, 2). We analyze the largest model output rank and centroid distance rank in replica partitions of each long-tailed data point, respectively. The hit rate with model output rank on data points v is set to 1 if $|S_{model}^v \cap S_{rep}^v| \neq \emptyset$, otherwise it is set to 0. The hit rate with centroid distance rank is calculated similarly by using S_{dist}^v , the top- M partitions in the centroid distance ranking of v . As shown in Fig. 4 (RIGHT), the model output rank can better indicate the replica partitions for achieving a higher hit rate at the same M .

3.4 Query-aware Top-k Retrieval

After model training and learning-based redundancy, we can achieve retrieval with the meta index alone or with two-level indexes (i.e., the internal indexes are required in large-scale datasets) [54]. We store the probing model and redundant partitions to evaluate the performance of *LIRA* as a meta index for partitions. In the following part of this section, we give a detailed description of top- k retrieval with a two-level index and regard an exhaustive search in a partition if using meta index alone. As shown in Fig. 6, the retrieval process includes two stages. In the first stage, we utilize the probing model as the meta index to get the probing probabilities as the retrieval guidance. In the second stage, we execute the searching in each probing partition with internal indexes.

We illustrate the retrieval process with *LIRA* as the meta index and HNSW as the internal index as an example. (1) In the first stage, we utilize the probing model to obtain retrieval guidance. Similar to the training process, we apply the trained model to query vectors and get the probing probabilities i.e., \hat{p}^q . Instead of probing a fixed number of partitions, *LIRA* supports adaptive $nprobe$ for each query with the predicted \hat{p}^q , where only those partitions with $\hat{p}^q > \sigma$ ($\sigma = 0.5$ for default) are treated as probing partitions. Hence, *LIRA* can prune more partitions and save more query fan-out compared

with a fixed $nprobe$. (2) In the second stage, *LIRA* executes the retrieval process with the probing partitions predicted in the first stage. When retrieving k results in a probing partition for query q , we use the internal indexes without an exhaustive search in the partition. After completing searches in all probing partitions, we merge all the retrieved data points as a coarse candidate set. Then, we rank the coarse candidate set according to the distance to the query and generate a precise candidate set as the top- k results.

4 Experiment

4.1 Experiment Settings

Datasets. We conduct experiments on 5 high-dimensional ANN benchmarks (See detailed description in Appendix A.4). Specifically, we evaluate *LIRA* on two small-scale datasets, SIFT [2] and GloVe [37]. We also show the scalability of *LIRA* on three large-scale datasets: Deep [4], BIGANN [24], and Yandex TI [50]. For the constraint in the RAM source, we subsample 50M data points for each large-scale dataset, following previous studies [18, 29].

Baselines. We evaluate *LIRA* as the meta index compared with four baselines, IVF in Faiss [25], IVFPQ [23], IVFFuzzy and BLISS [18] (See detailed description in Appendix A.4). Specifically, we build the IVFFuzzy index to show the effectiveness of redundancy through centroid distance rank, where every data point is placed in the two nearest clusters. BLISS is a learning-to-index method that builds four groups of partitions, each with an independent model.

To simulate the practical two-level kNN search, we build two-level indexes, *LIRA*-HNSW, and evaluate the effectiveness of *LIRA* as the meta index among partitions. We build two-level indexes for baselines similarly. To exclude the effect of the internal indexes, we first evaluate the effectiveness of *LIRA* and baselines as the meta index in small-scale datasets and then use two-level indexes in large-scale datasets. The detailed experiments of convergence validation and sensitivity analysis are in Appendix A.5 and A.6.

Settings. In *LIRA*, the number of partitions B is set as 64 and 1024 for small-scale and large-scale datasets, respectively. The k of kNN is mainly set as 100 since we focus on addressing probing waste and long-tailed kNN distribution. In index construction, the redundancy percentage η is set as 3% when using a meta index alone and is set as 100% when using a two-level index (See detailed sensitivity study on η in Appendix A.6). In query process, we use threshold σ to choose partitions with $\hat{p}^q > \sigma$ as probing partitions and tune σ from 0.1 to 1.0 with a step of 0.05. The number of partitions of baselines is the same as *LIRA*. For BLISS, we follow the original setting, build four groups of partitions with four independent models in the index construction phase, and search for four groups of partitions in the query phase.

Evaluation Metrics. For the one-level meta index, we evaluate the performance of *LIRA* and baselines threefold: accuracy, efficiency, and query fan-out. (1) We use Recall@ k to evaluate the search accuracy. (2) We use the distance computations cmp (i.e., the total number of visited data points) in the probing partitions to indicate the search efficiency. (3) We record the $nprobe$ to reflect the query fan-out and to reflect the effectiveness of partition pruning. For the two-level index in large-scale datasets, apart from the Recall@ k and $nprobe$, we additionally use query per second (QPS) to reflect the general search efficiency.

Table 1: Performance at Recall@ $k=0.98$ with various k .

<i>cmp</i>	IVF	IVFPQ	IVFFuzzy	BLISS	<i>LIRA</i>
$k = 10$	120641	120641 recall=0.70	119409 (-1.0%)	151911 (+25.9%)	83824 (-30.5%)
$k = 50$	137276	137276 recall=0.74	144120 (+4.9%)	151911 (+10.6%)	91431 (-33.3%)
$k = 100$	137276	137276 recall=0.76	144120 (+4.9%)	168778 (+22.9%)	96261 (-29.8%)
$k = 200$	153931	187410 recall=0.78	144120 (-6.3%)	168778 (+9.6%)	99279 (-35.5%)
<i>nprobe</i>	IVF	IVFPQ	IVFFuzzy	BLISS	<i>LIRA</i>
$k = 10$	7	7 recall=0.70	4 (-42.8%)	8 (+14.2%)	4.8138 (-31.2%)
$k = 50$	8	8 recall=0.74	5 (-37.5%)	8 (0%)	5.2342 (-34.5%)
$k = 100$	8	8 recall=0.76	5 (-37.5%)	9 (+12.5%)	5.4648 (-31.6%)
$k = 200$	9	11 recall=0.78	5 (-44.4%)	10 (+11.1%)	5.6561 (-37.1%)

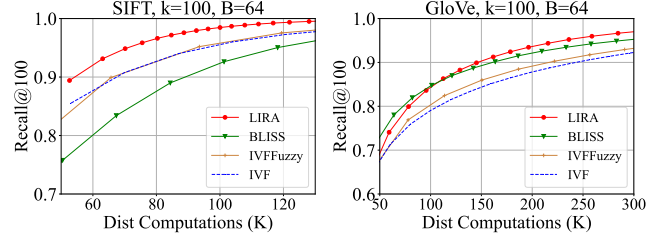
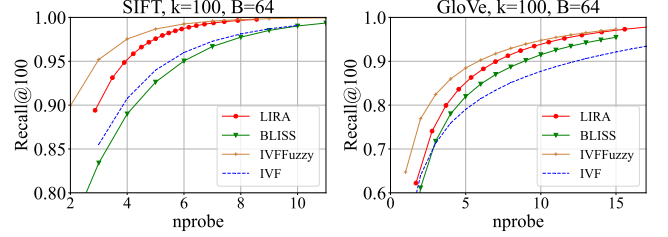
4.2 Evaluation on Small Scale Datasets

In this section, we first show the performance of *LIRA* and baselines when retrieving different k nearest neighbors and illustrate the superiority of *LIRA* with large k settings. Second, we show the trade-off between recall and distance computations and between recall and $nprobe$ on two small-scale datasets, SIFT and GloVe. For a fair comparison, we present the average distance computations executed by four individual models and partitions in BLISS.

Performance with Various k . To explore the performance on different retrieval requirements, we conduct experiments on SIFT with various k and calculate the minimum average distance computations and average $nprobe$ to achieve Recall@ $k = 0.98$. When IVFPQ can hardly achieve the desired recall, we record the acceptable distance computations with the corresponding recall value.

Table 1 shows that the *cmp* increases monotonically with k . Moreover, as k increases with more serious long-tailed kNN distributions, the advantages of *LIRA* are gradually highlighted. There are two reasons behind such an advantage. First, utilizing the strength of adaptive probing from a query-aware view, *LIRA* focuses on eliminating the probing waste by directly probing target kNN partitions with the model rather than probing according to distance ranks to centroids. Second, *LIRA* reduces the long-tailed kNN distributions with reasonable replicas. Table 1 also shows the required $nprobe$ for various k , which gives an apparent advantage of *LIRA* in saving query fan-out. When *LIRA* and other baselines except IVFFuzzy achieve the same recall, *LIRA* needs less $nprobe$. This result supports that the probing model in *LIRA* can prune partitions effectively and accurately. Due to the strong advantage of *LIRA* in a large k , we mainly present the results for Recall@100 in the rest of the experiments for brevity. IVFFuzzy is superior in reducing $nprobe$, but it performs comparably to IVF on *cmp*, for the average partition size in IVFFuzzy is double than that of other baselines. For the suboptimal performance of IVFPQ compared with IVF, we drop it from the remaining experiments.

Trade-off between Recall and both $nprobe$ and *cmp*. To manipulate the recall with corresponding *cmp*, we tune the $nprobe$ in IVF,

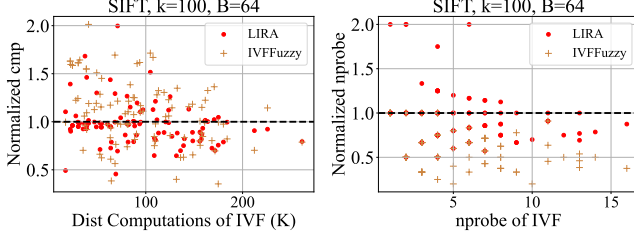
**Figure 7: Recall and *cmp* on small-scale datasets.****Figure 8: Recall and $nprobe$ on small-scale datasets.**

IVFFuzzy, and BLISS, and σ for partition pruning in *LIRA*. As shown in Fig 7 and Fig. 8, *LIRA* surpasses all the baselines in Recall@100. (1) Compared with other partition-based methods, *LIRA* outperforms for two reasons. First, the well-built redundant partitions in *LIRA* can naturally reduce the optimal $nprobe^*$ and the probing quantity for partitions with long-tail kNN . (2) Second, the probing model can better adaptively narrow down the area of the probing partition with the well-learned mapping from a data point in the vector space to the kNN partitions. (3) Moreover, the gap between *LIRA* and the baselines expands as the recall increases on GloVe. This is because the baselines struggle to tackle the phenomenon of the long-tailed kNN distribution, and the long-tail data points mainly impact the trade-off between recall and search cost at a high recall value. Hence, the advantage of *LIRA* is more outstanding with a high recall requirement through an effective query-aware partitioning pruning strategy. (4) As for performance on SIFT, when recall is higher than 0.98, the gap between *LIRA* and the baselines slightly shrinks. This is because *LIRA* does not fully address the long-tailed kNN distribution for considering efficiency, resulting in some long-tail data points without duplication.

It is also worth noting that the learning-based method, BLISS, performs much worse than other methods on SIFT. There are two main reasons for the inefficiency of BLISS. (1) BLISS builds partitions with models from scratch instead of refining partitions based on other clustering methods, e.g., the K-Means clustering algorithm. The partitions built in BLISS tend to be unbalanced without fine-tuned training, resulting in some partitions with a large number of data points while some other partitions have no data points. These unbalanced partitions may lead to suboptimal partition probing and cause more distance computations, *cmp*. (2) BLISS requires 4 groups of partition-based indexes. The $nprobe$ of BLISS is presented as the fixed $nprobe$ in one group of indexes, and the *cmp* of BLISS denotes the total number of candidates from four groups of partitions after deduplication. This also causes a waste of *cmp* since the four indexes built by independent models differ.

Table 2: Performance of QPS, Recall@100, partition pruning rate and overall latency on large-scale datasets.

Metrics	Query Per Second			Recall@100			<i>nprobe</i>		
	IVF	IVFFuzzy	<i>LIRA</i>	IVF	IVFFuzzy	<i>LIRA</i>	IVF	IVFFuzzy	<i>LIRA</i>
Deep	126	93	117	0.8996	0.9604	0.9655	9	11	9.2586
	227	338	354	0.8265	0.8354	0.8441	5	3	2.9567
BIGANN	62	104	118	0.9464	0.9607	0.9639	19	11	10.1043
	173	230	257	0.8496	0.8860	0.8952	7	5	4.6465
Yandex TI	141	160	174	0.7673	0.8316	0.8386	9	8	7.7058
	258	324	346	0.6847	0.7524	0.7732	5	4	3.8171

**Figure 9: Per query performance.**

Search Performance per Query. Fig. 9 presents the normalized distance computations *cmp* and *nprobe* of IVFFuzzy and *LIRA* over IVF on a per-query basis, respectively. Take the figure on the left of Fig. 9 as an example for a detailed explanation. The x-axis denotes the minimum *cmp* when IVF achieves the $Recall@k = 0.98$ for every query. The y-axis presents the ratio between the *cmp* of another method to achieve a recall of 0.98 and the *cmp* of IVF. A red point below the normalized value of 1.0 infers that the query process of this query with *LIRA* takes less retrieval cost than IVF, and a brown plus sign infers the normalized cost of IVFFuzzy. We remove the baseline BLISS from this experiment because it struggles to achieve the target recall efficiently. This figure provides the performance of individual queries, where we sample 100 queries from all the queries of the dataset for display.

Overall, IVFFuzzy and *LIRA* reduce the *cmp* compared with IVF at the same recall. For IVF, there are easy queries and hard queries, where hard queries require more probing partitions and distance computation to achieve the target recall. (1) It is worth mentioning in Fig. 9, *LIRA* optimizes most of the queries that need $nprobe \geq 10$ with IVF. This means that *LIRA* exhibits a significant reduction, especially on hard queries. This is because there are more long-tailed *k*NN distributions for hard queries, which results in more probing waste. (2) For the IVFFuzzy, it is desired to achieve half of IVF’s *nprobe* to achieve a comparable search efficiency in the one-level meta index, because the number of data points in IVFFuzzy is doubled. However, the general normalized *nprobe* of IVFFuzzy is more than 0.5. This illustrates that the redundancy strategy in IVFFuzzy is unable to achieve comparable search efficiency when IVFFuzzy is used as a meta index alone.

4.3 Evaluation on Large Scale Datasets

This section presents the performance of *LIRA* and other baselines on three large-scale datasets. We build two-level indexes with IVF,

IVFFuzzy and *LIRA* as the meta index, respectively. The HNSW index is used as the internal index for a fast inner-partition searching process. We drop BLISS from evaluation on large-scale datasets for out-of-memory with 4 groups of two-level indexes. Following previous study [18], Table 2 shows the performance in two scenarios that demand high efficiency or high recall, respectively. Based on the experiment results, we can make the following observations.

- Compared with non-learning methods, *LIRA* outperforms IVF and IVFFuzzy in most cases, especially with high recall.
- Compared to the performance of a one-level index, the IVFFuzzy becomes more efficient among the two-level indexes. With HNSW as the internal index, search efficiency is enhanced largely by avoiding exhaustive searches within a partition, and the large number of redundant data points has less impact on efficiency.
- Due to the learning-based redundancy and the query-aware adaptive *nprobe* generated by the effective probing model, *LIRA* can achieve better partition pruning than the IVF and IVFFuzzy.
- The improvement of *LIRA* compared to IVFFuzzy varies among different datasets, which may demonstrate that different η is required for different datasets. Hence, an opportunity exists for *LIRA* to achieve a better redundancy with an adaptive number of redundant data points on different datasets.

5 Conclusion

State-of-the-art partition-based ANN methods typically divide the dataset into partitions and use query-to-centroid distance rankings for search. However, they have limitations in probing waste and long-tailed *k*NN distribution across partitions, which adversely affects search accuracy and efficiency. To overcome these issues, we propose *LIRA*, a LearnIng-based queRy-aware parTition framework. Specifically, we propose a probing model to achieve outstanding partition pruning by reducing probing waste and providing query-dependent *nprobe*. Moreover, we introduce a learning-based redundancy strategy that utilizes the probing model to efficiently build redundant partitions, thereby mitigating the effects of long-tailed *k*NN distribution. Our proposed method exhibits superior performance compared with existing partition-based approaches in the accuracy, latency, and query fan-out trade-offs.

Acknowledgments

This work is partially supported by NSFC (No. 62472068, 62272086), Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021), and Municipal Government of Quzhou under Grant (No. 2023D044, 2023D004, 2023Z003, 2023D043, 2022D037, 2022D039, 2022D020).

References

- [1] Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore Willke. 2023. Similarity Search in the Blink of an Eye with Compressed Indices. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3433–3446.
- [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] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. Elpis: Graph-based similarity search for scalable data science. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1548–1559.
- [4] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2055–2063.
- [5] Patrick Chen, Wei-Cheng Chang, Jyun-Yu Jiang, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. 2023. Finger: Fast inference for graph-based approximate nearest neighbor search. In *Proceedings of the ACM Web Conference 2023*. 3225–3235.
- [6] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighbor search. *arXiv preprint arXiv:2111.08566* (2021).
- [7] Xu Chen, Haitian Chen, Zibo Liang, Shuncheng Liu, Jinghong Wang, Kai Zeng, Han Su, and Kai Zheng. 2023. Leon: A new framework for ml-aided query optimization. *Proceedings of the VLDB Endowment* 16, 9 (2023), 2261–2273.
- [8] Liwei Deng, Penghao Chen, Ximu Zeng, Tianfu Wang, Yan Zhao, and Kai Zheng. 2024. Efficient Data-aware Distance Comparison Operations for High-Dimensional Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2411.17229* (2024).
- [9] Liwei Deng, Yan Zhao, Jin Chen, Shuncheng Liu, Yuyang Xia, and Kai Zheng. 2024. Learning to hash for trajectory similarity computation and search. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4491–4503.
- [10] Liwei Deng, Yan Zhao, Zidan Fu, Hao Sun, Shuncheng Liu, and Kai Zheng. 2022. Efficient trajectory similarity computation with contrastive learning. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 365–374.
- [11] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. 2019. Learning space partitions for nearest neighbor search. *arXiv preprint arXiv:1901.08544* (2019).
- [12] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2022. Hercules Against Data Series Similarity Search. *Proceedings of the VLDB Endowment* (2022), 2005–2018.
- [13] Yuchen Fang, Yuxuan Liang, Bo Hui, Zezhi Shao, Liwei Deng, Xu Liu, Xinke Jiang, and Kai Zheng. 2024. Efficient Large-Scale Traffic Forecasting with Transformers: A Spatial Data Management Perspective. *arXiv preprint arXiv:2412.09972* (2024).
- [14] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2021), 4139–4150.
- [15] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.
- [16] 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.
- [17] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, et al. 2022. Manu: a cloud native vector database management system. *arXiv preprint arXiv:2206.13843* (2022).
- [18] Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J Smola. 2022. Bliss: A billion scale index using iterative re-partitioning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 486–495.
- [19] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.
- [20] Rui Hu, Yuze Wang, Ximu Zeng, Shuncheng Liu, Quanlin Yu, Peicong Wu, Zhengzhuo Zhang, Han Su, and Kai Zheng. 2024. Imitation learning decision with driving style tuning for personalized autonomous driving. In *International Conference on Database Systems for Advanced Applications*. Springer, 220–231.
- [21] 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.
- [22] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).
- [23] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.
- [24] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 861–864.
- [25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
- [26] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. 2020. Improving approximate nearest neighbor search through learned adaptive early termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2539–2554.
- [27] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. AI meets database: AI4DB and DB4AI. In *Proceedings of the 2021 International Conference on Management of Data*. 2859–2866.
- [28] Mingjie Li, Yuan-Gen Wang, Peng Zhang, Hanpin Wang, Lisheng Fan, Enxia Li, and Wei Wang. 2022. Deep learning for approximate nearest neighbour search: A survey and future directions. *IEEE Transactions on Knowledge and Data Engineering* 35, 9 (2022), 8997–9018.
- [29] Wuchao Li, Chao Feng, Defu Lian, Yuxin Xie, Haifeng Liu, Yong Ge, and Enhong Chen. 2023. Learning balanced tree indexes for large-scale vector retrieval. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1353–1362.
- [30] Defu Lian, Yongji Wu, Yong Ge, Xing Xie, and Enhong Chen. 2020. Geography-aware sequential location recommendation. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2009–2019.
- [31] Zibo Liang, Xu Chen, Yuyang Xia, Runfan Ye, Haitian Chen, Jiandong Xie, and Kai Zheng. 2024. DACE: A Database-Agnostic Cost Estimator. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4925–4937.
- [32] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 15, 2 (2021), 246–258.
- [33] Zepu Lu, Jin Chen, Defu Lian, Zaixi Zhang, Yong Ge, and Enhong Chen. 2024. Knowledge distillation for high dimensional search index. *Advances in Neural Information Processing Systems* 36 (2024).
- [34] Zepu Lu, Defu Lian, Jin Zhang, Zaixi Zhang, Chao Feng, Hao Wang, and Enhong Chen. 2023. Differentiable Optimized Product Quantization and Beyond. In *Proceedings of the ACM Web Conference 2023*. 3353–3363.
- [35] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.
- [36] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence* 36, 11 (2014), 2227–2240.
- [37] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.
- [38] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2012. Diversifying Top-K Results. *Proceedings of the VLDB Endowment* 5, 11 (2012).
- [39] Yongye Su, Yinqi Sun, Minjia Zhang, and Jianguo Wang. 2024. Vexless: A Serverless Vector Data Management System Using Cloud Functions. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–26.
- [40] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2023. DB-LSH 2.0: Locality-Sensitive Hashing With Query-Based Dynamic Bucketing. *IEEE Transactions on Knowledge and Data Engineering* (2023).
- [41] Hao Wang, Tong Xu, Qi Liu, Defu Lian, Enhong Chen, Dongfang Du, Han Wu, and Wen Su. 2019. MCNE: An end-to-end framework for learning multiple conditional network representations of social network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1064–1072.
- [42] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xianguo 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.
- [43] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2017. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2017), 769–790.
- [44] 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.
- [45] Ruijie Wang, Zheng Li, Danqing Zhang, Qingyu Yin, Tong Zhao, Bing Yin, and Tarek Abdelzaher. 2022. RETE: retrieval-enhanced temporal event forecasting on unified query product evolutionary graph. In *Proceedings of the ACM Web Conference 2022*. 462–472.
- [46] Xuguang Wang, Yiqun Lin, and Xiaomeng Li. 2023. Cgat: Center-guided adversarial training for deep hashing-based retrieval. In *Proceedings of the ACM web conference 2023*. 3268–3277.

- [47] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2406.10938* (2024).
- [48] Yuyang Xia, Shuncheng Liu, Quanlin Yu, Liwei Deng, You Zhang, Han Su, and Kai Zheng. 2024. Parameterized Decision-Making with Multi-Modality Perception for Autonomous Driving. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4463–4476.
- [49] Shitao Xiao, Zheng Liu, Weihao Han, Jianjin Zhang, Yingxia Shao, Defu Lian, Chaozhuo Li, Hao Sun, Denvy Deng, Liangjie Zhang, et al. 2022. Progressively optimized bi-granular document representation for scalable embedding based retrieval. In *Proceedings of the ACM Web Conference 2022*. 286–296.
- [50] Yandex. 2021. Text-to-Image-1B Dataset. <https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search>.
- [51] Renchi Yang. 2022. Efficient and effective similarity search over bipartite graphs. In *Proceedings of the ACM Web Conference 2022*. 308–318.
- [52] Ximu Zeng, Quanlin Yu, Shuncheng Liu, Yuyang Xia, Han Su, and Kai Zheng. 2023. Target-Oriented Maneuver Decision for Autonomous Vehicle: A Rule-Aided Reinforcement Learning Framework. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 3124–3133.
- [53] Kai Zhang, Chongyang Tao, Tao Shen, Can Xu, Xiubo Geng, Binxing Jiao, and Daxin Jiang. 2023. Led: Lexicon-enlightened dense retriever for large-scale retrieval. In *Proceedings of the ACM Web Conference 2023*. 3203–3213.
- [54] Pengcheng Zhang, Bin Yao, Chao Gao, Bin Wu, Xiao He, Feifei Li, Yuanfei Lu, Chaoqun Zhan, and Feilong Tang. 2023. Learning-based query optimization for multi-probe approximate nearest neighbor search. *The VLDB Journal* 32, 3 (2023), 623–645.
- [55] 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.
- [56] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1979–1991.
- [57] Bolong Zheng, Ziyang Yue, Qi Hu, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S Jensen. 2023. Learned probing cardinality estimation for high-dimensional approximate NN search. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 3209–3221.
- [58] Kai Zheng, Pui Cheong Fung, and Xiaofang Zhou. 2010. K-nearest neighbor search for fuzzy objects. In *Proceedings of the 2010 ACM SIGMOD international conference on Management of data*. 699–710.

A Appendix

In this section, we provide related work in Appendix A.1, motivation details in Appendix A.2, scalability analysis of *LIRA* in Appendix A.3, implementation details of experiments in Appendix A.4, convergence validation of the probing model training in Appendix A.5, and sensitivity analysis in Appendix A.6.

A.1 Related Work

Existing studies for ANN search can be roughly divided into four groups, including (1) hash-based [9, 21, 40], (2) tree-based [3, 12, 19, 36], (3) quantization-based [1, 15, 33], and (4) graph-based [14, 32, 35, 44]. Typically, the computational cost of retrieval k approximate nearest neighbors, $O(nd)$, incurs from the number of visited vectors denoted as n and the dimension of vectors represented as d . The existing ANN search methods leverage high-dimensional indexes to reduce latency from these two aspects.

Partition-based ANN Methods. The tree-based indexes [12] partition the vector space into nested nodes and then narrow down the search area with hierarchical tree-based indexes during the search phase. DB-LSH et al. [40] efficiently generate candidates by dynamically constructing query-based search areas. IVF (inverted file) index [25] first clusters the vectors into partitions and then narrows down the search area with the nearest partitions. Chen et al. [6] uses a clustering algorithm and the inverted index to build balanced posting lists, and probes the clusters within a certain distance from the query vector. Zhang et al. [54] focuses on multi-probe ANN search and formalizing the query-independent optimization as a knapsack problem. Neural LSH [11] generates partitions by balanced graph partitioning. Zhao et al. [56] combine the strength of LSH-based and graph-based methods and utilize LSH to provide a high-quality entry point for searching in graphs.

Learning-based ANN Methods. Artificial Intelligence(AI) has been widely applied to databases [7, 27, 31, 43], information retrieval systems [28] and decision-making scenarios [10, 13, 20, 48, 52]. We provide some learning-based ANN on top of partitions and graphs. For partition-based methods, BLISS [18] and Li et al. [29] combine the partition step and the learning step with learning-to-index methodology, and then search with a fixed $nprobe$. Zheng et al. [57] builds hierarchical balanced clusters and further leverages neural networks to generate adaptive $nprobe$ for each query. For graph-based methods, the graph-based indexes [35, 44] first connect the similar vectors with basic proximity graphs in the construction phase and then route in the graph through the most similar neighbors with the greedy search strategy. Based on the graph-based HNSW, Li et al. [26] demonstrate that easy queries often need less search depth than hard ones. Li et al. [26] introduces an early termination strategy and uses models to predict the minimum number of visited vectors required for retrieving the ground truth kNN for a given query, which can halt the search before meeting the traditional termination condition in the graph.

A.2 Motivation Details

We find that probing according to $nprobe_{dist}^*$ wastes probing cardinality, with empirical study presented in Fig. 10 (LEFT). The blue dashed line shows the percentage of queries with $nprobe^*$ no more than a specified $nprobe$. The red dashed line reflects the percentage of queries with $nprobe_{dist}^*$ no more than a given $nprobe$. In other

words, those queries achieving $Recall@10 = 1$ satisfy the accuracy requirement. (1) As we can see, the $nprobe_{dist}^*$ exceeds 20 for some queries when retrieving top-10 kNN , which introduces significant probing wastes. (2) Moreover, the waste of probing is even worse with a larger value of k . As shown in Fig. 10 (RIGHT) for $k = 100$, the $nprobe^*$ of all queries are no more than 22; while $nprobe_{dist}^*$ escalates to 40 for some queries.

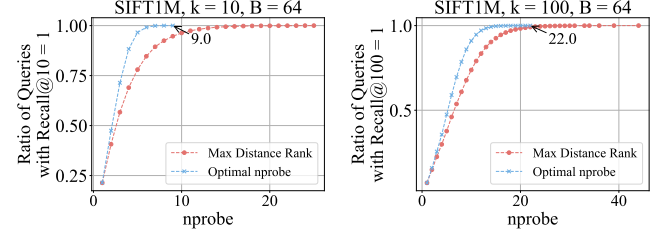


Figure 10: Probing Waste with Distance Ranking and Common Long-tail Phenomenon.

Algorithm 1 One-level and Two-level Index Building

- 1: Input: data points \mathcal{D} , training queries Q
 - 2: Output: indexes of \mathcal{D} in B partitions for top- k retrieval
 - 3: (1) For one-level meta index
 - 4: Sample a subset of data \mathcal{D}_{sub} for training ▸ Scalability
 - 5: Build B partitions for \mathcal{D}_{sub} and get partition centroids and distance to centroids I
 - 6: Get kNN partition distributions p of \mathcal{D}_{sub} as labels
 - 7: Learn $p^q, f(q, I) = p^q$ ▸ Model Training
 - 8: Put all \mathcal{D} in the nearest partitions
 - 9: Get the predicted probability \hat{p} of \mathcal{D} with $f(\cdot)$
 - 10: Pick $\eta\%$ of data points as \mathcal{D}_{pick} with \hat{p} ▸ Redundancy
 - 11: **for all** data point v in \mathcal{D}_{pick} **do**
 - 12: Choose a target replica partition with \hat{p}
 - 13: Duplicate the data of v
 - 14: **end for**
 - 15: (2) For two-level index
 - 16: Build internal indexes for each partition
-

A.3 Scalability Analysis

Since the subset of data is used in training, we give a detailed explanation of the scalability of *LIRA*. In general, *LIRA* only requires the ground truth kNN and kNN count distribution of a subset of data. The two phases of probing model training and learning-based redundancy are both scalable to large-scale datasets, even if the model is trained on a subset. The algorithm of index building with the probing model is illustrated in Algorithm 1.

First, for model training, the true label of kNN partition distribution is more sparse if we scale the number of data with the same total number of partitions B . This is because a partition contains more data points with a large-scale dataset under the same B , and the kNN of a query are more likely to be separated in fewer partitions. However, the true label of kNN partition distribution p

Table 3: Datasets

Dataset	# of Dimension	# of Data	# of Query
SIFT	128	1M	10K
GloVe	96	1M	1K
Deep	96	50M	10K
BIGANN	128	50M	10K
Yandex TI	200	50M	10K

does not affect the model training, since the probing partitions are selected by the threshold σ of the predicted possibilities.

Second, for learning-based redundancy, the picking and duplicating steps are all based on the relative results. We pick a data point v with a relatively high quantity of predicted probing partitions and then duplicate it to a partition with relatively high probing possibility in p^v . Hence, the workflow of *LIRA* is scalable to large-scale datasets.

A.4 Implementation Details

Datasets. We conduct experiments on 5 high-dimensional ANN benchmarks with different data sizes and distributions. The details of the datasets are shown in Table 3.

Baselines. The detailed information on baselines is as follows.

- IVF. IVFFlat in Faiss [25] (abbreviated as IVF) is a widely used ANN method that utilizes inverted indexes.
- IVFPQ. IVFPQ [23] is a widely adopted solution that combines the advantages of product quantization and the inverted file index.
- IVFFuzzy. Fuzzy clustering is a method where each data point can be put in more than one cluster. We build the IVFFuzzy index to show the effectiveness of redundancy through centroid distance rank, where every data point is placed in the two nearest clusters.
- BLISS [18]. BLISS is a learning-to-index method that builds groups of partitions, each with an independent model. The variant of BLISS [29] is omitted from experiments, as well as Neural LSH [11] that is inferior to BLISS.

For two-level indexes, the parameter of HNSW in graph building for limiting the edge of a data point is set as 32, and the search parameter of HNSW for limiting the length of the candidates set is set as 128.

Evaluation Platform. We implement our methods and baselines in Python 3.7. All the experiments are conducted on Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz, 256GB memory, and 4 NVIDIA GeForce RTX 3080.

A.5 Convergence Validation

This section is not intended to compare *LIRA* against other baselines but rather to verify the convergence of *LIRA*. We illustrate that during the process of model training and re-partition, the probing model can achieve convergence while the recall and probing fan-out can also be improved simultaneously. All the experiments on convergence validation are conducted on SIFT with 10K queries in the setting of $k = 100$ and $B = 64$. The batch size for model training is set as 512. To evaluate whether the predicted partitions are the k nn partitions, we also calculate the hit rate of k nn partitions.

Loss and Recall. As shown in Fig. 11(LEFT), we record the loss and the recall of the testing queries during the training process. A

step in the x-axis means every 10 batches of training data. We can observe that the loss in the blue line dramatically decreases and can finally achieve convergence. In addition, the recall in the red line decreases and then increases. This is because the positive label of target k NN partitions is sparse, and the probing model tends to predict few probing partitions at the beginning of training. With the probing model learning the mapping of data points to the target k NN partitions, the recall can approach nearly 1.0 at the end of the training.

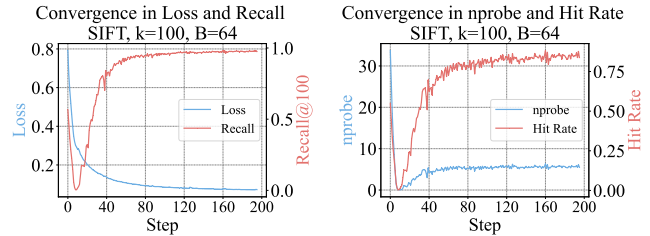


Figure 11: Model convergence validation with SIFT128, 1M.

***nprobe* and Hit Rate.** With the threshold of model outputs set as default $\sigma = 0.5$, we record the average *nprobe* of queries to represent the query fan-out and the hit rate of target probing partitions during the training process. As we can see in Fig. 11(RIGHT), the number of predicted *nprobe* of queries to represent the query fan-out and approximates the *nprobe**, and the hit rate in the red line can reach a high level. Hence, the result supports that the probing model can predict the target k NN partitions well. Even if the hit rate is about 0.8 after training, we can tune the threshold σ in the query process. In detail, a less σ results in more probing partitions and a larger *nprobe*, while a higher σ works in the opposite.

Scalability. We evaluate the scalability of *LIRA* by training on the subset of data and on the whole data, respectively. Specifically, for training on a subset, we sample D_{Sub} , 100K data from the 1M data in SIFT as the training data and building partitions on D_{Sub} . For $v \in D_{Sub}$, we get the k NN count distribution, n^v , and then get the k NN partition distribution p^v in D_{Sub} as training label. After model training, the learning-based redundancy is used on the whole data D . Keeping the partitions centroids unchanged, we put the whole data D in partitions and then use the probing model to achieve redundancy. For comparison, we also train a model with all the 1M data as training data.

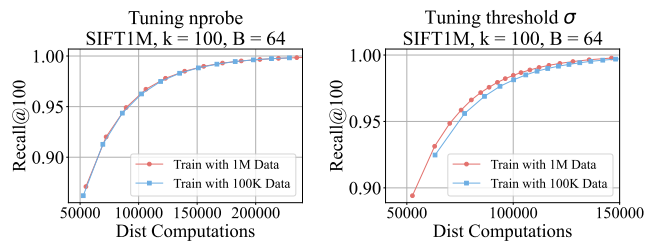
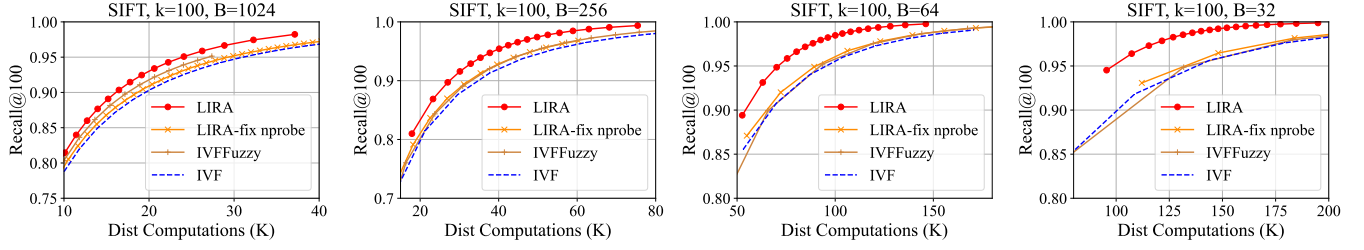
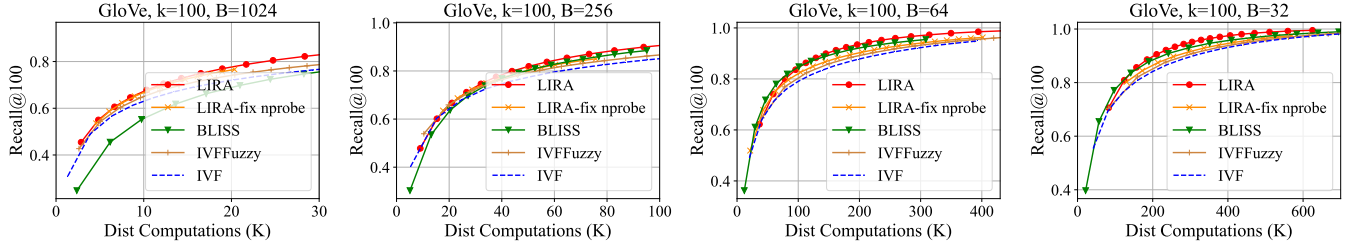


Figure 12: Scalability of model training.

Figure 13: Sensitivity of the number of partitions B in SIFT.Figure 14: Sensitivity of the number of partitions B in GloVe.

We get the trade-off between the recall and distance computations by tuning the $nprobe$ (i.e., probing partitions in the top $nprobe$ output rank) and by tuning the threshold σ (i.e., probing partitions with $\widehat{pq} > \sigma$). As shown in Fig. 12, the model trained on a subset performs similarly to the model trained on the whole dataset whether probing with $nprobe$ or threshold. The result supports that *LIRA* can be trained on a subset but can still achieve good partition redundancy and partition pruning on a whole dataset.

Time Cost Analysis. The time for partition construction in *LIRA* mainly comes from three aspects, training data collection, probing model training, and the internal index building. Following the experiment setting in this section, we first build the initial partition with K-Means and collect the kNN distributions of data, which takes 115 seconds. Second, we train the probing model and refine the partitions, which merely takes 108 seconds with the concise probing network. For small-scale datasets, training with six iterations can already achieve a good probing performance, although training iteration is set as ten as default. Third, we also build the internal index (i.e., HNSW) in each partition, which takes 12 seconds.

For the top- k retrieval process, we record the model inference time (i.e., to decide which partitions to search for a given query) and the total search time for all queries. We find that the model inference time only occupies less than 1% of the total search time, which shows the efficiency of predicting probing partitions. In detail, on SIFT with 10k queries, the time for getting the input (i.e., the distance calculation for queries to the centroids) is 0.2 second, the time for model inference merely takes 0.1 second, and the searching takes 238 seconds for the queries.

A.6 Sensitivity Analysis

Effect of B . The hyper-parameter B in *LIRA* is used as the total number of partitions. We conduct sensitivity analysis of the partition number B on SIFT and GloVe, which is presented in Fig. 13

and Fig. 14, respectively. To reflect the general trade-off between efficiency and accuracy, we plot these two figures with the average distance computations versus recall, comparing *LIRA-fix nprobe*, BLISS, IVFFuzzy, and IVF. The *LIRA-fix nprobe* is a variant of *LIRA* with different partition pruning, which utilizes a $nprobe$ configuration instead of adaptive $nprobe$ controlled by the threshold σ of the model output. We set the partition number as $B \in \{1024, 256, 64, 32\}$. Due to the inefficiency of BLISS on SIFT in Fig. 7 and Fig. 8, we exclude the plotting of this group of settings for the result unable to fit in the graph.

In all settings considered, *LIRA* outperformed other methods. (1) For a small value of B , *LIRA* can reduce distance computations since one partition contains more data points, and reducing one partition probing can save many data points from searching. (2) For a large value of B , *LIRA* can also decrease distance computations because the probing waste of baselines is more severe across a large number of partitions. (3) For *LIRA-fix nprobe*, even if this variant method searches with $nprobe$ configuration similar to the search process of IVF, it outperforms IVF consistently. This result supports that the probing model in *LIRA* can better directly probe the kNN partitions with model output rank than probe along with the centroid distance rank in IVF.

Effect of η . The hyper-parameter η in *LIRA* is used as the redundancy ratio, picking the data points with the highest predicted $nprobe$ for redundancy. Considering the internal index can reduce the adverse impact of redundancy of both *LIRA-HNSW* and IVFFuzzy-HNSW, we conduct sensitivity analysis of the redundancy ratio η on SIFT, which is presented in Fig. 15. We tune the η to duplicate different ratios of data points in *LIRA-HNSW*, while the IVFFuzzy-HNSW duplicates all data points. As we can see, merely duplicating 40% of data points in *LIRA-HNSW* can achieve comparable performance to IVFFuzzy-HNSW on the trade-off between recall and both distance computations and $nprobe$. For fair redundancy,

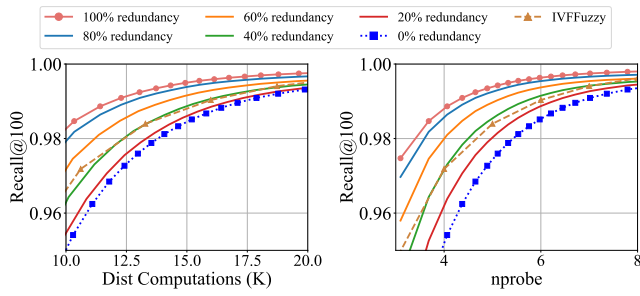


Figure 15: Sensitivity of the redundancy ratio η in SIFT.

we set the η as 100% in *LIRA*-HNSW as the same redundancy in IVFFuzzy-HNSW. When testing meta index alone without internal indexes, we set low redundancy $\eta = 3\%$ in *LIRA* to keep the high efficiency.