LIDAR: Learned Intrinsic Dimension-Aware Representations for Efficient Filter-and-Validation to Accelerate AKNN Search

Baohua Wu[†]
Central South University,
Changsha, China
bhwu.research@gmail.com

Yuhang Zhang[†]
Heilongjiang University
Heilongjiang, China
yhzhang.research@gmail.com

Lingli Li[*]
Heilongjiang University
Heilongjiang, China
lilingli@hlju.enu.cn

Zhifang Liao
Central South University
Changsha, China
zfliao@csu.edu.cn

Dongjing Miao
Harbin Institute of Technology
Harbin, China
miaodongjing@hit.edu.cn

ABSTRACT

The approximate K-Nearest Neighbor (AKNN) search in the high-dimensional spaces is a fundamental operation in many modern applications. Most existing methods for accelerating the AKNN search commonly follow the index-based filter-and-validation paradigm. To further accelerate this process, many methods attempt to find a low-dimensional AKNN-oriented representation from the original data representation to reduce the expensive distance computation cost during filter-and-validation. We observe that a much tighter AKNN-oriented representation can be obtained through a proper non-linear transformation. Therefore, in this paper, we propose a novel learning-based method named LIDAR (Learning Intrinsic Dimension-Aware Representation) training pipeline, which build a mapping function that transforms the original data representation to LIDAR, retaining only the information relevant to the AKNN search. Building upon LIDAR, we devise three generic techniques that can be seamlessly integrated as plugins into the existing AKNN query processing procedure. We demonstrate this integration method with two mainstream AKNN search methods. Theoretical analysis and extensive experiments on real-world datasets validate the effectiveness of our method. Compared to the original query processing procedure, LIDAR consistently improves the average end-to-end query time by up to 7.49× under the same recall.

PVLDB Reference Format:
Baohua Wu[†], Yuhang Zhang[†], Lingli Li[*], Zhifang Liao, and Dongjing Miao. LIDAR: Learned Intrinsic Dimension-Aware Representations for Efficient Filter-and-Validation to Accelerate AKNN Search. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

[†] Equal contribution, [*] Corresponding.

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

1 INTRODUCTION

In recent years, Approximate K-Nearest Neighbors (AKNN) search over high-dimensional embedded feature vectors has gained significant attention in the field of AI [24, 74], as it can serve as a cost-effective alternative to expensive model inference [14, 40, 41, 50, 67]. For example, Facebook employs it on embedded question and document vectors to facilitate semantic search [41]; Microsoft Bing has developed an efficient web-scale visual search system leveraging vector inverted files [40]. Specifically, given a set of n data objects $\mathcal{D} = \{o_1, \ldots, o_n\}$ where the representation[1] of each data object is a d-dimensional vector $o_i \in \mathbb{R}^d$, a query object $q \in \mathbb{R}^d$ and an integer $K \in \mathbb{N}^+$, the AKNN search aims to recall the K nearest neighbors in $\mathcal{D}$ of q as many as possible [23].

Existing methods widely used to accelerate AKNN search follow the index-based filter-and-validation paradigm [74], including clustering-based methods [5, 6, 16, 45], proximity graph (PG)-based methods [27, 58], locality-sensitive hash (LSH)-based methods [42, 72, 81] and so on. To avoid unnecessary expensive distance computations in original full-dimensional representation (denoted as FD-distance computations) during filter-and-validation, various methods expect to save computation by means of distance computations in an AKNN-oriented lower reduced-dimensional representation transformed from original representation (denoted as LD-distance computations), such as ADSampling [29], LSH-APG [80], PQ [45] and its variants [4, 30, 30, 32, 60]. Specifically, the LD-distances between the candidates and the query object are first computed, and only those candidates whose LD-distance does not satisfy the pruning criteria require expensive FD-distance computation to determine the final results. Even if these methods strive to reduce the cost of FD-distance computations, the sum cost of LD-distance and FD-distance computations still constitutes the dominant portion of the overall query processing time. Tab.1 reports the proportion of total query execution time spent on LD-distance and FD-distance computations for the aforementioned methods across four real-world datasets. The results show that the overall distance computations take over 70% of the total query time for all methods on most datasets. Therefore, *it is still crucial to significantly reduce the distance computation cost for an efficient AKNN search.*

[1]In this paper, we use *representation* to refer to either the vector of a single data object or that of a collection of data objects exchangeably when the context is clear.

Table 1: Proportion of distance computation time.

Dataset	PQ-IVF	IVF-Adsampling	LSH-APG
Sun	70.1 ± 3.4%	73.9 ± 2.6%	69.4 ± 3.1%
Enron	77.4 ± 2.7%	64.3 ± 1.9%	77.9 ± 2.8%
Glove	74.5 ± 3.1%	79.6 ± 2.9%	73.3 ± 3.4%
Msong	74.3 ± 2.5%	76.4 ± 3.1%	71.5 ± 2.7%

The above observation arises because once the approximation error between FD-distance and its corresponding LD-distance is too large to prune candidates up to a certain number, the LD-distance computations newly introduced result in a large time cost in addition, thus trading off the benefit gained from candidate pruning. Therefore, obtaining a much tighter AKNN-oriented representation, which decreases approximation error and dimensionality simultaneously as much as possible, is crucial for significantly lowering the overall cost of distance computations. All of the aforementioned methods (e.g., ADSampling, LSH-APG, PQ and their variants) leverage linear transformation to obtain the AKNN-oriented representations. This implies that a much tighter AKNN-oriented representation can be obtained by a non-linear transformation. This leads to the major challenge: *how to build a proper non-linear transformation rather than a linear one*, and *how to integrate it to the query processing procedure so as to accelerate AKNN search.*

To deal with this challenge, this paper studies the data-driven method to build a proper non-linear mapping function that transforms the original representation into a reduced-dimensional representation, retaining only the information relevant to AKNN search. The following insights figure out the necessary information which the desired non-linear mapping should preserve.

(I1) Dimensionality and information redundancy. A lower-dimensional representation is sufficient to preserve the information required for AKNN search due to the following two reasons: (i) The dimensionality of embedded feature vectors is chosen manually and is typically higher than necessary, in order to support efficient AI learning [31]. (ii) AKNN search requires only distance information between data objects, whereas high-dimensional vectors encode richer information. For example, in TransE embeddings [10], the vectors are subject to an algebraic constraint $h + r \approx t$, where h, r, and t denote the embedded vectors of the head entity, relation, and tail entity in a triple, respectively. Such a constraint, however, is irrelevant to AKNN search.

(I2) Non-uniform approximation requirement. For AKNN search, the tolerance for approximation error varies across the LD-distances between the query object and different data objects. In particular, only LD-distances between the query object and its proximal data objects whose FD-distances to the query object are small require accurate estimation, since large approximation errors would lead to a large number of candidates that require FD-distance computation. In contrast, as long as the approximate errors are not excessive, the LD-distances between the query object and its distant data objects are sufficient to identify that they are impossible to become the nearest neighbors. Fig.1 (a) presents the cumulative distribution function (CDF) of distances between query object and data objects in a real-world dataset *Sun*, showing that 95% of data objects can be pruned as long as their LD-distances to the query object are greater than 10^5 when $K \leq 10^3$.

(I3) Small local intrinsic dimensionality. Local intrinsic dimensionality (LID) quantifies the minimal dimensionality required to preserve distance information in the local neighborhood of a data object [38, 39], which is typically used to measure the difficulty of the AKNN search [56]. We observe that for a given data object, preserving local distance information with its proximal data objects requires only a small number of dimensions in real-world datasets. Fig.1 (b) presents the LID values computed for each data object w.r.t. its $1,000$ nearest neighbors in four datasets, showing that only 3.15%–11.19% of the original dimensions are sufficient to preserve local distance information. This insight, combined with **(I1)** and **(I2)**, implies that a low-dimensional representation focusing on local distance information preservation, while preserving global distance information as much as possible, is sufficient to accelerate AKNN search.

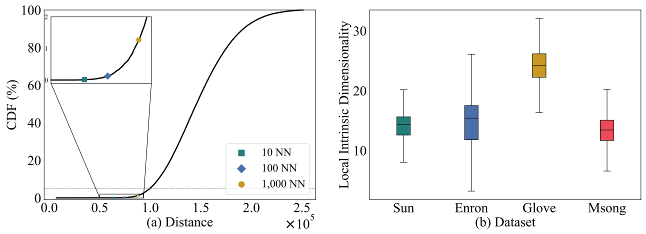

Figure 1: (a) Distance cumulative distribution function between query object and data objects on *Sun*. 🟩, ♦, and ● represent 10, 100 and $1,000$ NN, respectively. The gray dashed line represents 5%. (b) Distribution of the LID for all data objects w.r.t. their $1,000$ nearest neighbors across four datasets. The original dimensionality of these datasets are 512, $1,369$, 300 and 420, respectively.

Motivated by this, we propose a novel contrastive learning-based method called LIDAR (Learned Intrinsic Dimension-Aware Representations) training pipeline, which learns the mapping from the original representation through the universal approximation capability of neural networks [37]. To our best knowledge, this is also the first study to obtain the AKNN-oriented representation by learning a non-linear mapping function. Building upon the LIDAR, we develop generic techniques that can seamlessly integrate it as a plugin into existing AKNN search methods, in order to improve the efficiency of filter-and-validation. The main contributions of this paper are listed as follows.

Firstly, we propose the LIDAR training pipeline, which efficiently learns the mapping from the original representation to the low-dimensional representation tailored for AKNN search. The loss function and the contrastive sample pair construction method are carefully designed for the LIDAR training pipeline. The contrastive loss function mitigates the impact of distant contrastive sample pairs on neural network parameter updates. The contrastive sample pairs are dynamically selected in each epoch by means of an in-batch construction trick to allow LIDAR preserve distance information efficiently and effectively.

Secondly, we show the way to accelerate the filtering and validating via LIDAR. We replace FD-distance computation with

LD-distance during index-based filtering, thus yielding a deterministic decrease in distance computation cost. We empirically verify that filtering on LIDAR identifies a tighter search scope, since the LID of LIDAR is smaller compared to that of the original representation. We leverage LD-distance to prune candidates during validating.

Thirdly, in terms of algorithmic implementations, we devise two kinds of cache-friendly vector storage layout to support most existing AKNN search methods. Proper data layout improves the efficiency of the single distance computation due to increased cache utilization.

Finally, extensive experimental results demonstrate the effectiveness and efficiency of the proposed techniques. For example, when the recall of KNN is 95%, LIDAR brings an end-to-end speedup of 4.37-21.28× and 1.53-7.29× over IVF and HNSW, respectively. Ablation studies further validate the effectiveness of each component of our method.

The rest of this paper is organized as follows. Section 2 describes the LIDAR training pipeline, including the training architecture, the loss function, and the training set construction method. The integration method which integrate LIDAR into the existing AKNN search process is proposed in Section 3. Section 4 reports the experimental results. We review the related work in Section 5 and conclude this paper in Section 6.

2 LIDAR TRAINING PIPELINE

In this section, we present our training method, which learns the mapping from the original representation to LIDAR. Fig.2 provides the LIDAR training framework. We present the model architecture of this training pipeline in Section 2.1. We provide a formal definition of the LIDAR loss function in Section 2.2. We describe the contrastive sample pair construction method in Section 2.3.

2.1 The Model Architecture

As illustrated in Fig.2, LIDAR training framework adopts the Siamese network architecture for training [12], because this architecture ensures that similar inputs are still mapped to nearby points in the transformed space [17]. This architecture consists of two parameter-sharing subnetworks (a.k.a. twin networks). Specifically, it accepts distinct inputs that are joined by an energy function at the top. This function measures the difference in distance between the transformed representation and the original representation, which the training process aims to minimize. For the subnetwork (i.e., the mapping function[2] $f(\cdot)\colon \mathbb{R}^d \mapsto \mathbb{R}^{d'}$ with $d' \ll d$), we adopt a fully connected 3-layer perceptron, with each hidden layer employing the activation function $ReLU(\cdot)$. The size of each linear layer decreases progressively with depth, enabling the mapping function to reduce dimensionality in multiple stages. Compared to constant-width architectures, a progressively decreasing dimensionality has been verified to extract essential features more effectively [31]. This mapping function can be formulated as follows:

$$f(o) = W_3^\top \cdot ReLU(W_2^\top \cdot ReLU(W_1^\top \cdot o)), \tag{1}$$

[2]In this paper, we use *mapping function* and *model* exchangeably when the context is clear.

where W_i denotes the learnable parameter of the i-th linear layer. We denote the learned representation from the original representation $\mathcal{D}$ as $f(\mathcal{D})$.

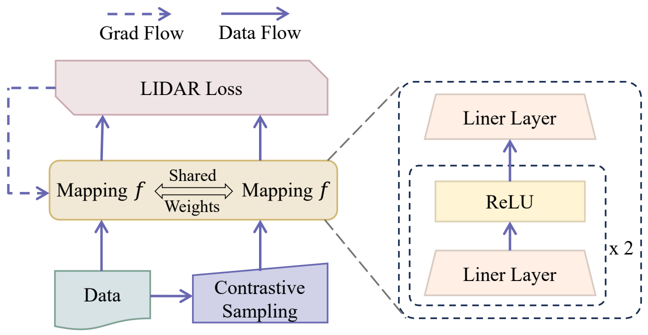

Figure 2: The LIDAR training framework.

Theorem 1 states that the above model can preserve distance relationships between arbitrary objects. Although other network architectures can also satisfy this property [33], their inference time is typically higher than that of multilayer perceptron.

Theorem 1. *The mapping function $f(\cdot)$ is Lipschitz continuous with constant $\rho = \prod_{i=1}^{3} \|W_i\|$. Specifically, for any two data objects o and q, the approximation ratio between their LD-distances and FD-distance can be bound as:*

$$\frac{\|f(o), f(q)\|}{\|o, q\|} \le \rho, \tag{2}$$

where $\|W_i\|$ denote the Spectral norm of the matrix W_i.

Proof (Sketch). For the function $h(x) = W_i^\top \cdot x$, it is *Lipschitz continuous* with *constant* $\|W_i\|$. For the function $ReLU(x) = max(x, 0)$, it is *Lipschitz continuous* with *constant* 1. According to the properties of *Lipschitz* function composition, we obtain that *Lipschitz constant* of $f(\cdot)$ is $\|W_1\| \cdot \|W_2\| \cdot \|W_3\|$. Detail proof can be found in the Appendix. □

2.2 The Loss Function

An intuitive idea for learning distance information is to minimize the mean squared error (MSE) between the LD-distances and the corresponding FD-distances of contrastive sample pairs. However, this loss function contradicts our objective, because it tends to focus on minimizing the approximate errors of the distant contrastive sample pairs rather than the proximal contrastive sample pairs. Specifically, distant contrastive sample pairs lead to larger gradients and thus contribute more to optimization during gradient descent [11, 48, 63]. The quadratic characteristic of MSE will cause the loss of distant contrastive sample pairs disproportionately larger than proximal contrastive sample pairs, even when their distance ratios between LD-distances and FD-distances are identical, and Example 1 illustrates this situation in detail.

Example 1. *Consider a data object o and its two contrastive objects u and v. Suppose the FD-distances are $\|o, u\| = 10$ and $\|o, v\| = 100$, while the corresponding LD-distances in the transformed space are $\|f(o), f(u)\| = 8$ and $\|f(o), f(v)\| = 80$. In both cases, their distance* *ratios are identical:* $\frac{8}{10} = \frac{80}{100} = 0.8$. *However, their MSE contributions differ significantly: the squared error for object* p *is* $(8 - 10)^2 = 4$, *while for object* q *it is* $(80 - 100)^2 = 400$.

To mitigate this issue, we introduce an additional loss term based on the distance ratio between the LD-distance and FD-distance. Specifically, we adopt the squared logarithmic error of this ratio, which naturally penalizes deviations from 1 and reduces the influence of the absolute distance difference. The LIDAR loss function is defined as:

$$\begin{aligned}\mathcal{J}(u,v) =& \lambda \cdot (\|f(u), f(v)\| - \|u,v\|)^2 + \\ & (1-\lambda) \cdot log^2(\frac{\|f(u), f(v)\|}{\|u,v\|}),\end{aligned} \tag{3}$$

where $\lambda \in [0,1]$ denotes the hyperparameter that balances the contributions of the absolute error and the ratio term, and (u, v) is the contrastive sample pair to be sampled and learned.

2.3 Contrastive Sample Pair Construction

The distances of the contrastive sample pairs encapsulate the information that $f(\cdot)$ is intended to preserve [13]. If we enumerate the distances between all possible object pairs in $\mathcal{D}$, this will lead to the following two issues: (i) the total number of training samples is $O(n^2)$, which is computationally infeasible for a large-scale dataset; and (ii) the number of distant contrastive sample pairs is much greater than that of proximal contrastive sample pairs, which leads to the former dominating the loss.

To address issue (i), we train the model on a subset $\mathcal{D}'$ of size n', uniformly sampled from $\mathcal{D}$, where n' is a hyperparameter. To address issue (ii), we increase the learning frequency of proximal contrastive sample pairs. Specifically, for each $o_i \in \mathcal{D}'$ throughout the entire training process, its local contrastive objects remain fixed, while its global contrastive objects are dynamically selected in each epoch. We select top-k nearest neighbors in $\mathcal{D}'$ of o_i as its local contrastive object set, denoted as $\mathcal{P}(o_i)$, where k is a hyperparameter that controls the local region size. For the global contrastive object set, we aggregate the local contrastive sets of all other data objects in a mini-batch to represent it. Formally, for $o_i \in \mathcal{B}$, its global contrastive object set is defined as $\mathcal{N}(o_i) = \cup_{o \in \mathcal{B} \setminus \{o_i\}} \mathcal{P}(o)$, where $\mathcal{B}$ denotes a mini-batch uniformly sampled from $\mathcal{D}'$ without replacement in each epoch. Therefore, the complete contrastive set of o_i is the union set of the local contrastive sets of all data objects in $\mathcal{B}$. Clearly, this is also the complete contrastive set for all objects in $\mathcal{B}$.

For the contrastive sample pair construction process described above, we formulate the learning objective for the mapping function $f(\cdot)$ at each epoch as follows:

$$\arg\min_{W_i} \sum_{\mathcal{B} \subset \mathcal{D}'} \sum_{o_i \in \mathcal{B}} \sum_{o_j \in \mathcal{A}_{\mathcal{B}}} \mathcal{J}(o_i, o_j), \tag{4}$$

where $\mathcal{A}_{\mathcal{B}} = \cup_{o \in \mathcal{B}} \mathcal{P}(o)$ denotes the contrastive set for the batch $\mathcal{B}$. Compared to training on a fixed contrastive object set, our method can offer the following advantages: (1) For each object, its global contrastive objects vary across epochs, allowing the model to capture global distance information instead of being limited to a static subset. (2) For each object, its local contrastive objects are explicitly selected in each epoch, while its global contrastive objects are selected uniformly at random. The probability of the global contrastive object being sampled repeatedly over multiple epochs decreases exponentially. This mitigates the overemphasis on specific contrastive sample pairs during gradient updates, which emphasizes the learning of proximal contrastive sample pairs. (3) Since all objects in $\mathcal{B}$ share the same contrastive object set, fewer vectors are loaded into the GPU memory compared to assigning an individual contrastive object set for each data object. We provide a detail space complexity analysis and elaborate on the training process in the Appendix.

Remark. In data-parallel training, the standard approach splits the mini-batch $\mathcal{B}$ across multiple GPUs, computes the loss and gradients independently on each device, and then aggregates the gradients to update the model parameters. However, in our training strategy, the data objects within a mini-batch are interdependent. Therefore, this standard approach will result in gradients that deviate from those computed in standard (single device) training. To address this issue, we recommend partitioning $\mathcal{A}_{\mathcal{B}}$ across multiple GPUs and replicating $\mathcal{B}$ entirely on each GPU. The losses and gradients are independently computed on each GPU, after which the gradients are aggregated to update the model parameters.

3 LIDAR APPLICATION

In this section, we provide the method that integrates LIDAR with existing AKNN search methods to accelerate filter-and-validation. We review the existing AKNN query processing procedures in Section 3.1. We discuss how to leverage LD-distances to reduce the distance computation cost in Section 3.2. In addition, we summarize the general implementation of optimization techniques as plugins to enhance the existing AKNN algorithm in Section 3.3.

3.1 AKNN Search Overview

In this paper, we focus on the following two important categories of the existing practical AKNN search [79]: **(i) two-stage search**: the complete dataset is first filtered based on index lookup, and then the candidate set is validated based on the FD-distance computation. The indexes that support two-stage search, include clustering-based and partially LSH-based indexes [46, 72]; and **(ii) iterative search**: a candidate set is initialized based on the index lookup and then iteratively updated and validated by expanding the search frontier. The indexes that support iterative search, include PG-based, the remaining LSH-based [42] and hybrid-based indexes [76]. In the following, we introduce these query processing procedures in detail.

Two-Stage AKNN search. Algorithm 1 illustrates the pseudo-code of a two-stage AKNN search. Initially, the candidates of q are retrieved by index $\mathbb{I}_{\mathcal{D}}$ lookup (line 3). Subsequently, for each candidate object o, the distance between o and q is computed, and the current top-K nearest neighbors of q are recorded as result set (lines 4–6). The search radius r is the distance between q and the K-th nearest objects during the search.

Iterative AKNN search. Algorithm 2 illustrates the pseudo-code of iterative AKNN search. Initially, one or several objects close to q in $\mathcal{D}$ are retrieved by index $\mathbb{I}_{\mathcal{D}}$ lookup (line 3) as candidates. Subsequently, the search frontier is expanded by retrieving the neighbors of current candidates, and the distances between q and the unvisited candidates are computed to update the candidate set

Algorithm 1: Two-stage AKNN search

```
Input: Index $\mathbb{I}_{\mathcal{D}}$, Query object $q$, Number of neighbors $K$,
       Search scope control parameters $c$.
Output: List of top-$K$ nearest neighbors.
1 $\mathcal{R} \leftarrow$ empty max-heap                ▷ Initialize result set
2 $r \leftarrow +\infty$                                  ▷ search radius
3 $C \leftarrow$ indexLookup($\mathbb{I}_{\mathcal{D}}$, $q$, $c$)
4 for $o$ in $C$ do
5     if $\|o, q\| \leq r$ then
6         update $\mathcal{R}$ and $r$
7 return $\mathcal{R}$
```

(line 7 – 9) until the termination condition is reached (line 4). The top-s nearest neighbors in these candidates of q are recorded as result set (line 9). The search radius r is the distance between q and the s-th nearest objects during the search, where $s \geq K$ to ensure search accuracy.

Algorithm 2: Iterative AKNN search

```
Input: Index $\mathbb{I}_{\mathcal{D}}$, Query object $q$, Number of neighbors $K$,
       Search scope control parameters $c$, Result set size $sz$.
Output: List of top-$K$ nearest neighbors.
1 $C$, $\mathcal{R} \leftarrow$ empty min-heap and max-heap
2 $r \leftarrow +\infty$                                  ▷ search radius
3 Initialize $C$, $\mathcal{R}$ through indexLookup($\mathbb{I}_{\mathcal{D}}$, $q$, $c$)
4 while termination conditions $\neq$ True do
5     $u \leftarrow C$.top()
6     $C$.pop()
7     for $v$ in indexLookup($\mathbb{I}_{\mathcal{D}}$, $u$, $c$) do
8         if $v$ not visited and $\|v, q\| < r$ then
9             update $C$, $\mathcal{R}$, and $r$
10 return top-$K$ nearest neighbors in $\mathcal{R}$
```

In both query processing procedures, the major distance computation cost arises from validating the result set, i.e., lines 5 in Algorithm 1 and line 8 in Algorithm 2. The remaining distance computation cost arises from filtering the candidate set from $\mathcal{D}$, i.e., line 3 in Algorithm 1 and lines 3, 7 in Algorithm 2. Although the number of distance computations in the filter stage is much smaller than that in the verification stage, the results of the filter stage determine the number of distance computations in the verification stage. We will discuss how to apply LD-distance to reduce the distance computation cost in both the filter and verification stages in the next subsection.

3.2 AKNN Search Acceleration

Intuitively, we can replace all FD-distance computations with LD-distance computations to achieve the maximum performance improvement. However, this method can hurt the search accuracy badly without the correction of FD-distances, because the approximation errors could reorder the distance-based ranks of proximal objects for query object. Therefore, we propose the following strategies to reduce distance computation cost by introducing negligible accuracy loss: **(1) filtering acceleration:** replacing all the FD-distance computations with LD-distance computations in the filter stage (Section 3.2.1); **(2) validating acceleration:** first pruning the candidates with LD-distances and only the candidates that do not satisfy prune criteria needs to be verified through FD-distance computations in the validation stage (Section 3.2.2); and **(3) layout refinement:** devising the corresponding data layout for these two AKNN query processing procedures to reduce cache misses, improving the efficiency of the single distance computation (Section 3.2.3).

3.2.1 ***Filtering acceleration.*** The filter stage is a coarse-grained search that aims to find as close a subset or object in $\mathcal{D}'$ to a reference object as possible based on index lookup. Since this operation does not need to obtain accurate results, we propose to leverage LD-distances instead of FD-distances during the filter stage. Specifically, we leverage the learned representation of the original dataset $f(\mathcal{D})$ to build an index $\mathbb{I}_{f(\mathcal{D})}$ instead of $\mathbb{I}_{\mathcal{D}}$, and perform the lookups based on this index.

Discussion. For the number of dimensions involved in the distance computations, our method yields a deterministic decrease by a factor of $\frac{d'}{d}$ compared to the original method. As an example, consider the index IVF, the time complexity of index $\mathbb{I}_{\mathcal{D}}$ lookup takes $O(N_c \cdot d)$, while that in $\mathbb{I}_{f(\mathcal{D})}$ is $O(N_c \cdot d')$, where N_c is a constant. However, this naturally raises a key question: *Does relying solely on LD-distances compromise the accuracy?* Recall that our goal is to eliminate irrelevant features of high-dimensional vectors, which do not contribute to the AKNN search. We observe that this process increases the discriminability of the data. Specifically, in our experiments, the LID values decrease 58.6% on average and the approximation ratio for 99% of the distances is between 0.9 and 1.1. This improved discriminability is achieved without significantly altering pairwise distances, which we believe facilitates a more efficient index lookup. We report more detailed results in Section 4.3.2.

3.2.2 ***Validating acceleration.*** For the validation stage in both Algorithms 1 and 2, a candidate object o contributes to the result if it satisfies the threshold condition: $\|o, q\| \leq r$. If the LD-distance $\|f(o), f(q)\|$ is sufficient to determine whether o satisfies the condition, the FD-distance computation can be avoided. Therefore, we introduce an extra condition before the FD-distance computation during the validation stage. We first compute $\|f(o), f(q)\|$ and validate whether it exceeds r. If so, o is pruned. Otherwise, we compute the FD-distance $\|o, q\|$. However, as shown in Theorem 1, $\|f(o), f(q)\|$ is not always smaller than $\|o, q\|$. Consequently, there are cases where $\|o, q\| < r$ and $\|f(o), f(q)\| > r$, leading to incorrect pruning of the contributing objects. To mitigate this issue, we introduce a distance confidence coefficient α to refine the pruning criteria. Specifically, the candidate object o is pruned only if $\|f(o), f(q)\| > \alpha \cdot r$.

Discussion. Now, we discuss the setting of α. As α increases, the incorrect pruning ratio gradually decreases to 0 until α reaches the *Lipschitz constant* of $f(\cdot)$. Although proximal objects to the query object require a larger α than distant objects, we fix α as a constant. This is because the distance-based ranks of data objects w.r.t. the query object cannot be predetermined, and α has little impact on distant objects since their distances to the query object are much larger than r. In our experiments, we observe that setting $\alpha = 1.1$ is sufficient to avoid 99% incorrect pruning while decreasing the pruning ratio by no more than 5%.

Analysis. Next, we analyze the performance improvement of distance computation. Let $\bar{d}$ and $\hat{d}$ denote the number of dimensions involved in the distance computations during the validation stage in the original query process and the pruned query process, respectively. Clearly, we have $\bar{d} = |C| \cdot d$. For $\hat{d}$, it consists of three parts: (1) transform q to $f(q)$; (2) compute the LD-distance between $f(q)$ and the entire candidate set; (3) compute the FD-distance between q and the pruned candidate set. Hence, the expression of $\hat{d}$ is:

$$\hat{d} = d' \cdot d + |C| \cdot d' + (1-p) \cdot |C| \cdot d, \tag{5}$$

where p denotes the proportion of candidate objects that are successfully pruned. The following corollary presents the condition that the distance computation cost is decreased.

Corollary 1. *The distance computation cost of the pruned query process procedure is lower than that of its original query process procedure when* $p > \frac{d'}{|C|} + \frac{d'}{d}$.

In our experiments, we observe that $\frac{d'}{|C|} + \frac{d'}{d}$ remains below 30%, while the average pruning success ratio p is 83.85%. This empirically verifies the effectiveness of our method.

3.2.3 ***Layout refinement****.* In our preliminary experiments, we observe that the computation cost of the single distance computation in the iterative search is nearly twice that in the two-stage search. This discrepancy arises because the two-stage search directly obtains the complete candidate set, which allows the index to store sequentially the objects that will become candidates at the same time. In contrast, the iterative search cannot predict the next object involved in the distance computation, leading to random storage accesses during filtering and validating. Therefore, the integration of LIDAR and original representations to support efficient filter-and-validation introduces the following two issues: (i) If we store LIDAR randomly, a large number of random storage accesses will be incurred during the query processing procedure. (ii) For the two-stage search, pruning candidate objects based on LD-distances will disrupt this sequential access pattern.

To address these issues, we devise the corresponding storage layouts between LIDAR and the original representation for these AKNN search methods, to fully exploit cache performance. Notably, different storage layouts only modify the data location in memory or disk without affecting the search results.

For the iterative search, we propose the *Coupled Dimension Storage (CDS)* layout to avoid the cache miss penalty. Specifically, we concatenate the LIDAR with its original representation for each object and store the concatenate representation following the original storage layout. Fig.3 provides an illustrative example of *CDS*. For a candidate object in the query process procedure, we first access its learned representation to compute its LD-distance to query object. Subsequently, if this LD-distance does not satisfy pruning criteria, we access its original representation sequentially to compute its FD-distance to query object, which eliminates the cache miss penalty. The *CDS* layout for iterative search ensures that random accesses do not increase compared to the original index design.

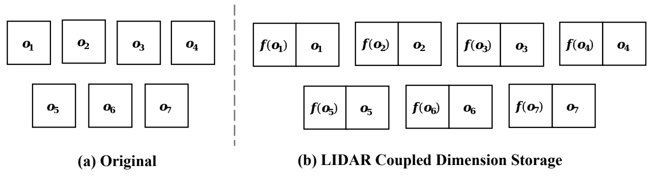

Figure 3: Example of Coupled Dimension Storage Layout.

For the two-stage search, we propose *Decoupled Dimension Storage (DDS)* layout to avoid random memory accesses caused by pruning. Specifically, we concatenate the LIDAR and the original representations separately while maintaining the same storage order between them, and store them sequentially. Fig.4 provides an illustrative example of *DDS*. For each set of objects that need to be validated sequentially in the query process procedure, we first compute their LD-distances to query object sequentially and record these LD-distances. Subsequently, we access only the original representations of objects that require FD-distance computations and reuse the stored LD-distances for secondary filter before FD-distance computations, because the search radius r gradually decreases during this process.

Analysis. We analyze the number of random storage accesses caused by pruning under the *CDS* and *DDS* layouts, denoted m_c and m_d, respectively. Multiple sets of sequentially stored objects will be accessed in the query process procedure. Since the overall result is the accumulation of results in individual sets, we focus on the scenario where only a single set is accessed sequentially. We assume that the size of this set is m. Since pruning a candidate object directly leads to random access in the *CDS* layout, the m_c is $p \cdot m$. For m_d, the minimum occurs when all pruned objects are placed at the end of the set, whereas the maximum occurs when the retained objects are completely non-adjacent in the storage. The range of m_d is $[0, min(p \cdot m,\ (1-p) \cdot m)]$. Thus, we have the following corollary.

Corollary 2. *For the two-stage search, the number of random storage accesses under the CDS layout is equal to that under the DDS layout only in the worst-case scenario.*

In our experiments, we observe that m_d rarely reaches its maximum value. To validate our observations, we further perform a Permutation Test [26] on the access sequence. These results confirm that the randomness of storage access is statistically significant at the 95% confidence level. Detail experimental results can be found in the Appendix.

For an AKNN method that integrates all the above optimizations, we refer to it as AKNN-LIDAR. For example, we denote the optimized version of IVF as IVF-LIDAR.

3.3 AKNN-LIDAR Search Summary

In this subsection, we integrate the above optimizations into the AKNN search, and propose two enhanced AKNN search algorithms: two-stage AKNN-LIDAR search and iterative LIDER-AKNN search.

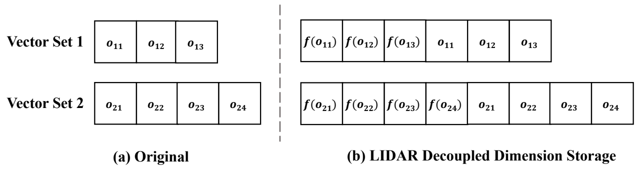
Vector Set 1
o_{11} o_{12} o_{13}
Vector Set 2
o_{21} o_{22} o_{23} o_{24}
(a) Original
$f(o_{11})$ $f(o_{12})$ $f(o_{13})$ o_{11} o_{12} o_{13}
$f(o_{21})$ $f(o_{22})$ $f(o_{23})$ $f(o_{24})$ o_{21} o_{22} o_{23} o_{24}
(b) LIDAR Decoupled Dimension Storage

Figure 4: Example of Decoupled Dimension Storage Layout.

Compared to the original AKNN search method, the differences in the index construction stage of AKNN-LIDAR are leverage $f(\mathcal{D})$ to construct and store $f(\mathcal{D})$ and $\mathcal{D}$ as described in Section 3.2.3. In the following, we summarize these AKNN-LIDAR query processing procedures in detail.

Two-stage AKNN-LIDAR search. Algorithm 3 illustrates the two-stage AKNN-LIDAR search pseudo-code. Initially, the candidates of $f(q)$ are retrieved by index $\mathbb{I}_{f(\mathcal{D})}$ lookup (line 3). Subsequently, all sequentially stored vector sets in the candidate set are iterate access to record the final result (lines 4-14). Specifically, for each vector set, the LD-distances between candidate objects and the query object are first computed, and then the candidate objects that are not pruned and the corresponding LD-distances are recorded as the filtered candidate set C' (lines 5-9). Next, for each object of filtered candidate set (i.e. $o \in C'$), the LD-distance $\|f(o), f(q)\|$ is leveraged to secondary pruning (lines 11-12), and the FD-distance $\|o, q\|$ will be computed only if $\|f(o), f(q)\| < \alpha \cdot r$. The top-$K$ nearest neighbors of q in the candidate set is recorded during this process (line 14). We provide a more specific example using IVF [45] in the Appendix.

Algorithm 3: Two-stage AKNN-LIDAR search

```
Input: Index $\mathbb{I}_{f(\mathcal{D})}$, Query vector $q$, Number of neighbors
       $K$, Search scope control parameters $c$, Distance
       confidence coefficient $\alpha$.
Output: List of top-$K$ nearest neighbors.
1  $\mathcal{R} \leftarrow$ empty max-heap                ▷ Initialize result set
2  $r \leftarrow +\infty$                                  ▷ search radius
3  $C \leftarrow indexLookup(\mathbb{I}_{f(\mathcal{D})}, f(q), c)$
4  for $\mathcal{S}$ in $C$ do
5      $C' \leftarrow \emptyset$                          ▷ Filtered candidate set
6      for $o$ in $\mathcal{S}$ do
7          if $\|f(o), f(q)\| > \alpha \cdot r$ then
8              continue     ▷ Filtering non-contribute candidate
9          $C' \leftarrow C' \cup \{\|f(o), f(q)\|, o\}$
10     for $dist$, $o$ in $C'$ do
11         if $dist > \alpha \cdot r$ then
12             continue             ▷ Secondary filtering candidate
13         if $\|o, q\| \leq r$ then
14             update $\mathcal{R}$ and $r$
15 return $\mathcal{R}$
```

Iterative AKNN-LIDAR search. Algorithm 4 illustrates the pseudo-code of iterative AKNN-LIDAR search. Initially, one or several objects that are as close to $f(q)$ are retrieved by index $\mathbb{I}_{f(\mathcal{D})}$ lookup (line 3). Subsequently, the neighbors of the current candidate object are continuously selected to expend the search frontier (line 7). For each expanded candidate object v, the LD-distance $\|f(o), f(v)\|$ is first computed for pruning (lines 8-9). If $\|f(v), f(q)\| < \alpha \cdot r$, the original representation of v will be accessed sequentially to compute the FD-distance $\|v, q\|$ for updating the result set (lines 13-14). We provide a more specific example using HNSW [58] in the Appendix.

Algorithm 4: Iterative AKNN-LIDAR search

```
Input: Index $\mathbb{I}_{f(\mathcal{D})}$, Query vector $q$, Number of neighbors
       $K$, Search scope control parameters $c$, Result set size
       $sz$, Distance confidence coefficient $\alpha$.
Output: List of top-$K$ nearest neighbors.
1  $C$, $\mathcal{R} \leftarrow$ empty min-heap and max-heap
2  $r \leftarrow +\infty$                                   ▷ search radius
3  Initialize $C$, $\mathcal{R}$ through $indexLookup(\mathbb{I}_{f(\mathcal{D})}, f(q), c)$
4  while termination conditions $\neq$ True do
5      $u \leftarrow C$.top()
6      $C$.pop()
7      for $v$ in $indexLookup(\mathbb{I}_{f(\mathcal{D})}, f(u), c)$ do
8          if $v$ not visited and $\|f(v), f(q)\| > \alpha \cdot r$ then
9              continue     ▷ Filtering non-contribute candidate
10         if $\|v, q\| < r$ then
11             update $C$, $\mathcal{R}$, and $r$
12 return top-$K$ nearest neighbors in $\mathcal{R}$
```

4 EXPERIMENTS

4.1 Experimental Setting

Platform and Implementation. All experiments are performed on a workstation running 64-bit Ubuntu 22.04 LTS, equipped with 8 Intel Xeon E7-8860@2.2GHz, 3TB DRAM and 4 NVIDIA RTX A6000. To avoid NUMA effects, we use the numactl utility to bind each thread to the necessary number of NUMA nodes. We implement the LIDAR training pipeline in C++, with the mapping function $f(\cdot)$ trained on the GPU and the inference performed on the CPU by libTorch [64]. We manually implement data parallelism to facilitate parallel training. In this paper, we focus on two mainstream indexes to verify the end-to-end performance of LIDAR: IVF (clustering-based method) and HNSW (PG-based method). We implemented IVF-LIDAR based on *faiss* [21, 47] and HNSW-LIDAR based on *hnswlib* [62]. We have also open-sourced the parameter weights of the mapping function on Google Drive. All codes are compiled using GCC v11.4.0 with -O3. Regarding the hyperparameter, we perform a grid search to enumerate possible combinations across all algorithms and present the results corresponding to the best overall performance. We also open-source the results of running all algorithms under all parameters in the repository so that subsequent researchers could reproduce them. For detailed parameter configurations, please refer to the experimental setting table. All models are trained with the same seed.

DataSets and Query Workload. We evaluate the performance of our algorithm and the comparison algorithms on nine real-world high dimensional datasets, which cover various applications, including audio, text, and image. Their main characteristics are summarized in Tab. 2. All download links can be found in our open source repository. Following standard practice, we randomly selected $n_q = 1,000$ data objects from each dataset and added random Gaussian noise to generate query objects. Notably, the query objects are not involved in any training process. We set $K = 100$ according to practical needs.

Table 2: Statistics of Datasets.

Dataset	Number (n)	Dim (d)	LID	Size	Category
Sun	79,106	512	13.96	154.5 MB	image
Enron	94,987	1,369	14.39	496.05 MB	text
Nuswide	269,648	500	22.51	514.31 MB	image
Glove	400,000	300	24.04	457.76 MB	text
Msong	994,185	420	13.25	1.56 GB	audio
Word2vec	1,000,000	300	22.48	1.12 GB	text
Gist	1,000,000	960	20.36	3.58 GB	image
Tiny40M	40,000,000	384	18.31	56.72 GB	image
Tiny80M	80,000,000	384	19.63	113.44 GB	image

Competitors. To demonstrate the effectiveness of LIDAR, we compare HNSW-LIDAR and IVF-LIDAR with their original version, i.e., HNSW and IVF. To demonstrate the effectiveness of our LD-distance computation, we compare it with another SOTA LD-distance computation method, ADSampling [29], which dynamically estimates distances by hypothesis test to reduce unnecessary distance computation cost during vector scanning. We refer to Adsampling optimized IVF and HNSW as IVF-Adsampling and HNSW-Adsampling. We also compare it with two mainstream quantization methods: Product Quantization (PQ [45]) and Optimized Product Quantization (OPQ [30]). To ensure search recall, we only leverage quantized distance to pruning, similar to Section 3.2.2, since we found in experiments that this is usually more efficient than re-rank. Additionally, to further validate the performance of our method, we compare them with two other PG-based indexes: NSSG [27] and LSH-APG [80]. NSSG adopts sophisticated graph construction strategies to prune a KNN graph [20], preventing excessive graph density that could degrade search efficiency. LSH-APG leverages LSH to guide entry points into the proximity graph and leverages the distance in the hash space to prune candidate objects.

Evaluation Metric. We employ two most important metrics to evaluate the end-to-end performance of these algorithms. (1) Recall: which measures the proportion of correctly retrieved ground truth K nearest neighbors to K; and (2) Query time (QT): defined as the average CPU time per query. To evaluate the effectiveness of the mapping function, we further introduce the following additional metrics. (3) False negative rate (FNR): the proportion of data objects that are not included in the result set and do not satisfy the pruning criteria, relative to the size of the candidate set; and (4) False positive rate (FPR): the proportion of incorrectly pruned objects, computed as the difference between the original result set and the pruned result set, normalized by the size of the original result set. Formally, the FPR is: $\frac{|\mathcal{R}\backslash\mathcal{R}^*|}{|\mathcal{R}^*|}$, where $\mathcal{R}^*$ denotes the original result set and $\mathcal{R}$ denotes the pruned result set.

Due to space limitations, more experimental results are provided in the Appendix, including the index construction performance, the number of nearest neighbors, and the randomness of storage access.

4.2 Overall Performance

In this subsection, we compare our methods, IVF-LIDAR and HNSW-LIDAR, with eight competitors across nine datasets and show the trade-offs between query time and recall in Fig.5 (bottom-right is better). We summarize the key findings as follows. Notably, due to the prohibitive time cost of PG-based index construction, several algorithms failed to build indexes on certain datasets, e.g., all PG-based indexes cannot be constructed in one day for the largest dataset *Tiny*.

(1) *Comparison with the ablation baselines.* From Figs.5 (a)-(i), both IVF-LIDAR and HNSW-LIDAR demonstrate substantial performance improvements against the original indexes. More specifically, IVF-LIDAR improves efficiency by up to 10.2× on average compared to IVF, and HNSW-LIDAR improves efficiency by up to 3.5× on average compared to HNSW. In addition, we also observe that the efficiency improvement becomes more significant as the recall increases. This result indicates that the more candidates are retrieved for higher recall, the more distance computation cost can be reduced by our method.

(2) *Comparison with the SOTA LD-distance computation methods.* The results in Figs.5 (a)-(i) indicate that AKNN-LIDAR achieves a substantial performance improvement over all competing methods designed for improving the distance computations, including ADSampling, PQ, and OPQ. Notably, while the other methods exhibit similar performance, LIDAR consistently demonstrates superior efficiency, achieving an average speedup of up to 2.4×. These experimental results validate the superiority of our non-linear transformation over the linear transformation methods, which enables accurate LD-distance estimation for complex data distributions at a lower computational cost.

(3) *Comparison with the PG-based competitors.* From Figs5 (a)-(i), both our methods beat all the PG-based baselines across all datasets regarding the *query time vs. recall* trade-off. Compared to the SOTA PG-based competitors, i.e., LSH-APG and NSSG, HNSW-LIDAR achieves a speedup of up to 2.9× and 9.2×, respectively, under the same recall. This is because the search in the PG-based indexes typically identifies smaller candidate sets than that in the clustering-based indexes. In addition, IVF-LIDAR also comprehensively outperforms these competitors in multiple datasets (2.5× on average), including *Sun*, *Word2vec*, and *Msong*. This improvement is attributed to our ability to prune at least 85% of the candidates by computing the LD-distances at low cost.

In summary, LIDAR-AKNN exhibits superior query performance than all competitors. Moreover, compared to the existing LD-distance computation methods, LIDAR can bring the best efficiency improvement for the existing AKNN index.

4.3 Ablation Study

In this subsection, we focus on evaluating the effectiveness of the components we propose. Notably, we only report the results on four representative datasets (with high LID values and multiple

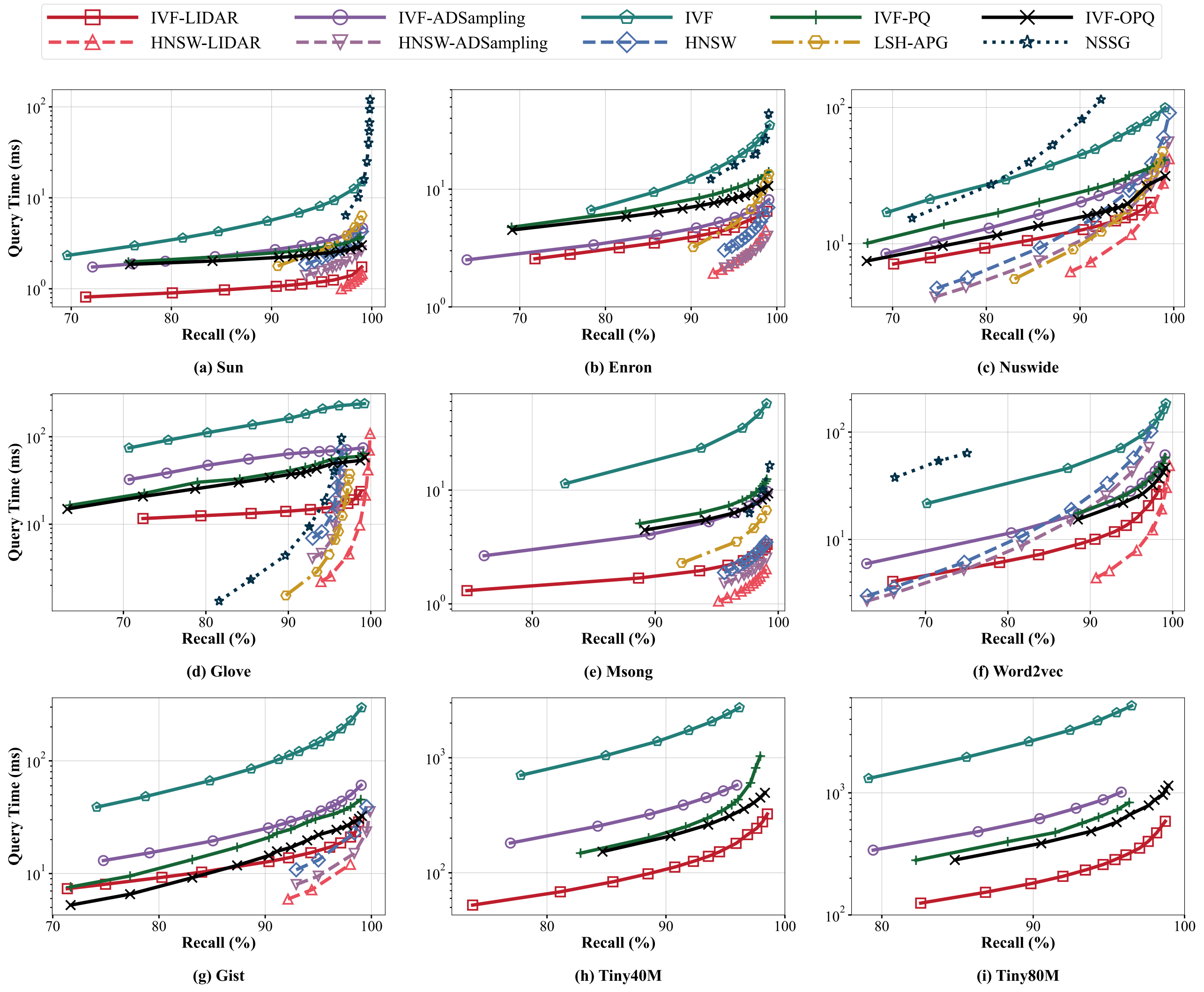

Figure 5: QT-Recall curves among all competitors (bottom-right is better).

categories) for the sake of brevity, including *Sun, Nuswide, Glove, and Msong*.

4.3.1 ***Effectiveness of sampling.*** To study the effectiveness of our contrastive sample pair construction approach, we compare it with its two variants: (1) *CompleteSamp*, as described in Section 2.3; (2) *LocalSamp*, where we select only the top-1000 nearest objects as comparison objects; and (3) *GlobalSamp*, where we uniformly sample an equal number of contrastive objects from $\mathcal{D}'$ in each batch. Fig.6 illustrates the curve between query time and recall, and the average FNR and FPR. From Figs.6 (a)–(d), we observe that the *CompleteSamp* outperforms *GlobalSampl* (1.4× on average) in all datasets and outperforms *LocalSamp* (2.3× on average) in all datasets except HNSW in *Sun*. The reason for this special case on *Sun* is that k = 1,000 fully covers the candidate set of HNSW in this scenario. From Figs.6 (e)–(f), we further observe that *CompleteSamp* sampling achieves the best trade-off among different scenarios. Although *LocalSamp* and *Only Global* can decrease the FNR in some cases (*Sun* and *Nuswide*), they also increase the FPR, which trades off their benefits.

4.3.2 ***Effectiveness of Index Lookup.*** To illustrate the effectiveness of index lookup via LD-distances, we compare our methods with the ablation index (i.e., the index constructed on original representation) in terms of the average number of candidates to reach 95% recall. The results are depicted in Tab.3 (left). To our surprise, our low-dimensional index, AKNN-LIDAR, yields a smaller candidate set size than its original version, resulting in an average reduction of 25.5%.

For example, on *Glove*, IVF-LIDAR filters a smaller candidate set than IVF for all queries, which decreases the size 66.2% on average. This phenomenon can be attributed to the alleviation of the "curse of dimensionality", since the learned representation focuses more on local distance relationships, and the distance distribution

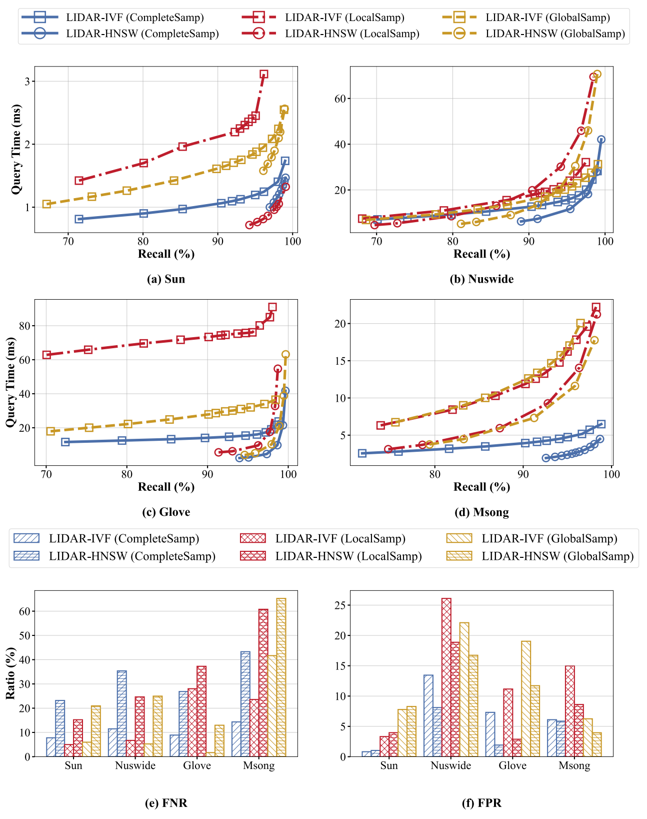

Figure 6: Performance of AKNN-LIDAR with different contrastive sample pair construction approaches.

between the learned representations is more uniform from a global perspective. To validate this observation, we further perform a Wilcoxon signed-rank test [77] between the LIDAR results and the original results. Tab.3 (right) illustrates the statistic W^- and the corresponding p-value. Here, W^- measures how much better the LIDAR results are than the original ones, with a theoretical maximum of $\frac{n_q \cdot (n_q+1)}{2} = 500,500$. These results confirm that the improvement of the lookup effect for AKNN-LIDAR compared to its original version is statistically significant at the 95% confidence level.

Table 3: Performance of index lookup via FD-distances and LD-distances.

Dataset	Index	Candidate Set Size		Wilcoxon	
		w/o LIDAR	w/ LIDAR	W^-	p-value
Sun	IVF	8,511	7,703	496,140	7.17e-160
	HNSW	2594	2452	421,235	3.82e-79
Nuswide	IVF	65,890	52,097	465,026	1.61e-122
	HNSW	18,716	13,870	431,635	9.24e-89
Glove	IVF	322,150	109,082	500,500	1.66e-165
	HNSW	16,601	5,125	500,261	3.41e-165
Msong	IVF	23,164	21,975	369,918	1.36e-39
	HNSW	1,384	1,364	321,918	2.48e-16

4.3.3 ***Effectiveness of storage layout.*** To evaluate the effect of storage layout, we report the average query time of IVF-LIDAR and HNSW-LIDAR under different layouts (*CDS* and *DDS*) in Tab.4. We observe that, *DDS* yields better performance than *CDS* (improved by 69% on average) for IVF, whereas *CDS* performs better than *DDS* (improved by 19% on average) for HNSW. These results align well with our hypothesis in Section 3.2.3, i.e., a proper storage layout can improve the storage access efficiency in distance computation. We also observe that layout refinement brings more improvements to IVF than to HNSW. The reason is that IVF search is a two-stage search while HNSW search is an iterative search, thus IVF has more continuous storage access requirements than HNSW.

Table 4: The average query time (ms) of IVF-LIDAR and HNSW-LIDAR between different storage layouts.

Dataset	Index	Data Layout		Improvement (%)
		CDS	DDS	
Sun	IVF-LIDAR	1.625	1.194	36.09
	HNSW-LIDAR	0.999	1.149	14.93
Nuswide	IVF-LIDAR	23.28	16.28	42.97
	HNSW-LIDAR	11.69	14.46	23.64
Glove	IVF-LIDAR	26.75	15.54	72.11
	HNSW-LIDAR	2.559	3.015	17.81
Msong	IVF-LIDAR	4.943	2.187	125.9
	HNSW-LIDAR	1.135	1.379	21.53

4.3.4 ***Effectiveness of the secondary filter.*** To verify the effectiveness of the secondary filter in two-stage AKNN-LIDAR search, we compare the average query time of IVF-LIDAR with and without the secondary filter. As shown in Tab. 5, IVF-LIDAR with the secondary filter consistently outperforms its counterpart across all datasets, achieving an average improvement of 5.6% in query efficiency. This is due to the fact that more candidate objects can be pruned by the secondary filter as the search radius decreases during the query processing procedure.

Table 5: The average query time (ms) of IVF-LIDAR with and without the secondary filter.

Dataset	Sun	Enron	Nuswide	Glove	Msong
w/ secondary filter	1.298	4.886	16.39	15.72	2.299
w/o secondary filter	1.324	4.944	16.83	16.34	2.591
Improvement (%)	2.022	1.201	2.669	3.878	12.67

4.4 Parameter Sensitivity Analysis

In this section, we investigate the impact of all hyperparameters during model training. All FNR and FPR are reported in the same search scope under the IVF index.

4.4.1 ***Effect of the number of layers.*** To investigate the impact of network depth, we vary the number of layers within the range of {1, 3, 5, 7, 9, 11} and report both the FNR and the FPR in Fig. 7. As shown in Fig. 7(a) and (b), the model performance initially improves as the number of layers increases, but deteriorates rapidly beyond a certain point. This is because greater depth enhances the model's representational capacity but also impedes gradient flow, making it difficult to optimize parameters in the early layers.

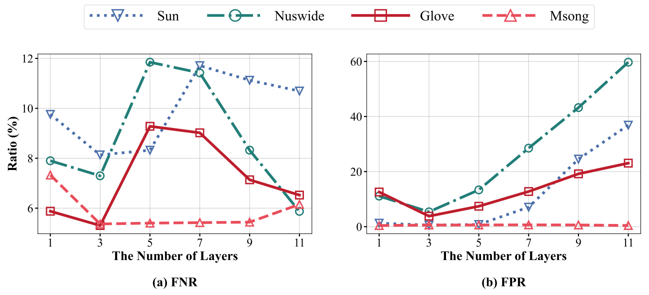

Figure 7: Parameter Study on the number of layers.

4.4.2 *Effect of the intrinsic dimension.*

We evaluate the impact of the intrinsic dimension d' on both the FNR and the FPR by varying d' from $0.05d$ to $0.3d$. The results are shown in Fig. 8. As expected, both the FNR and FPR decrease monotonically as d' increases. This trend is intuitive, since a higher intrinsic dimension preserves more information, leading to more accurate distance estimation.

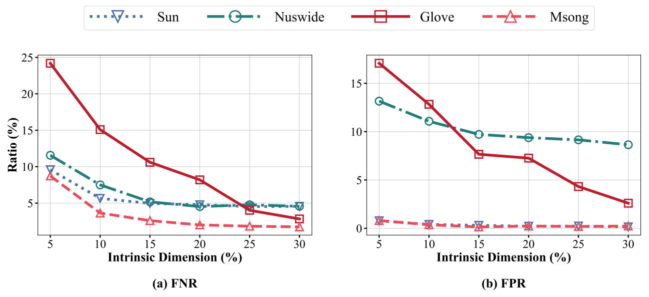

Figure 8: Parameter Study on d' of LIDAR training pipeline.

4.4.3 *Effect of k.*

We investigate the impact of the number of local contrastive objects per data object by varying this number within the range of {10, 30, 50, 70, 100}. Fig. 9 illustrates how this parameter affects the model performance. We observe that both FNR and FPR generally decrease as k increases and becomes stable after k is large enough. Moreover, we observe that FNR and FPR in *Nuswide* and *Glove* reach stability earlier. This phenomenon occurs because the data objects in *Nuswide* and *Glove* are more indistinguishable than those in *Sun* and *Msong*, i.e., the nearest neighbors of query object are closely spaced in them, so they can use fewer local contrastive objects to figure out the local region necessary for AKNN search. Therefore, during training, the value of k should be selected according to the specific data distribution and query requirements.

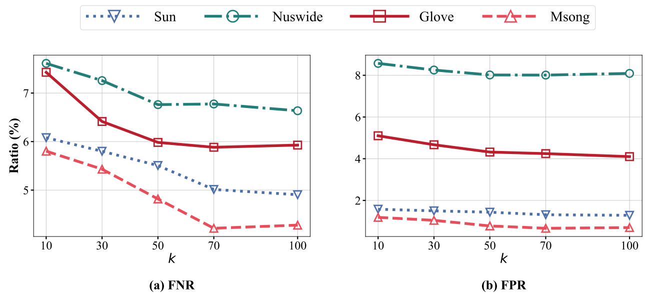

Figure 9: Parameter Study on k.

4.4.4 *Effect of λ.*

We investigate the impact of λ by varying it within the range of {0, 0.2, 0.4, 0.6, 0.8, 1}. As shown in Fig. 10 (a) and (b), increasing λ initially decreases the FNR but eventually causes it to rise again, while the FPR exhibits a monotonic upward trend. This behavior occurs because larger λ values improve the emphasis on learning global distance information while reducing the focus on local distance information, which can lead to false negatives during result validation. Moreover, employing only the MSE loss (i.e., $\lambda = 1$) yields the poorest performance, as the distance ratio loss is essential for narrowing the gap between proximal and distant contrastive pairs. Conversely, setting $\lambda = 0$ to use the distance ratio loss alone is also suboptimal, as it fails to adequately emphasize distant contrastive pairs, compromising the accuracy of their LD-distance estimates and thus the pruning effectiveness.

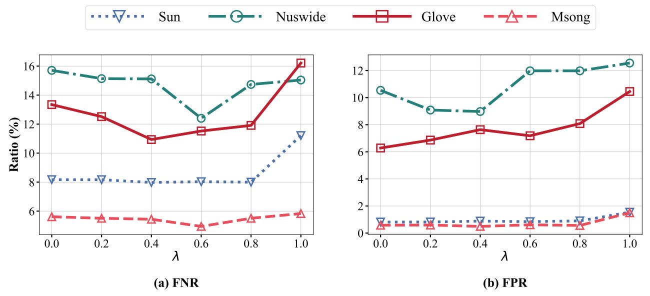

Figure 10: Parameter Study on λ.

4.4.5 *Effect of n'.*

To evaluate the impact of training set size, we train the mapping function on a series of nested subsets from the *Tiny80M* dataset. Specifically, we first uniformly sample 10 million objects as our largest subset, then progressively construct smaller subsets of {1M, 3M, 5M, 7M} objects, ensuring each is a proper subset of the next larger one. We evaluate the pruning performance of these models on both the 10M and 80M datasets, as shown in Fig. 11. The results reveal consistent trends across both datasets: model performance improves and eventually saturates with more training data. Furthermore, each model attains a lower FNR on the larger dataset, which we attribute to the higher proportion of distant objects per query. This observation aligns with the second insight discussed in Section 1. Interestingly, using the entire training set does not yield noticeable improvements compared to using partial data (e.g., 7M *vs.* 10M). These findings suggest that retraining is unnecessary in practice, as long as the data distribution remains relatively stable.

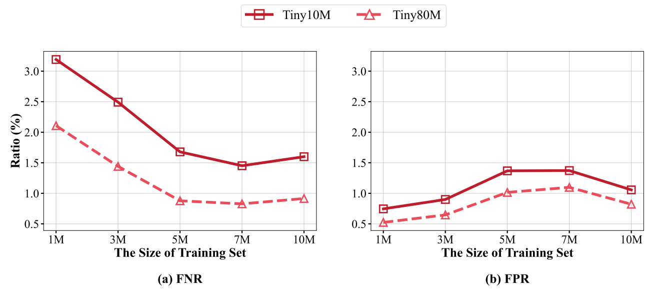

Figure 11: Parameter Study on n'.

5 RELATED WORK

5.1 AKNN Indexes and Optimization.

5.1.1 ***AKNN Indexes.*** The existing AKNN indexes can be broadly categorized into three structural categories: (1) Tree-based indexes[9, 18, 34, 49, 66, 73]; (2) Clustering-based indexes [5, 6, 16, 28, 45]; and (3) PG-based indexes [27, 44, 58, 59, 75, 75]. Since tree-based indexes exhibit substantially inferior performance compared to PG-based and cluster-based approaches on high-dimensional datasets, we do not focus on them in this work. They are now commonly used as auxiliary structures for other AKNN search methods [57, 72]. For example, DB-LSH [72] leverages R-trees [34] to perform range queries to identify candidate sets. Mircosoft STAPG [15] leverages KD-tree [8] and Balanced K-Means Tree [2] to improve the routing in the proximity graph. While PG-based indexes typically achieve the best query performance among AKNN methods, their graph construction is significantly slower—sometimes by hundreds of times—compared to clustering-based and LSH-based approaches [80]. LSH-based indexes, in contrast, allow for fast construction and provide theoretical guarantees [19, 43, 57, 69]. However, as widely reported [3, 56, 80], these methods can be multiple times slower than PG-based and clustering-based indexes when achieving the same recall. As a result, clustering-based and PG-based indexes are the most practical solutions. To address the issue of graph construction on large-scale datasets, many methods first partition the data via clustering, then construct graphs within each cluster, and finally merge them into the final graph [61, 76].

5.1.2 ***Distance Computation Optimization.*** Andre et al. [1] transforms cache-resident PQ lookup tables into small tables that fit within Single Instruction Multiple Data (SIMD) registers, which decreases cache misses and enables efficient SIMD computation. However, their design is limited to an 8-bit quantization setting with $m = 8$, which is insufficient for modern high dimensional data. ADSampling [29] leverages a random orthogonal matrix to transform the original data during index construction. During the query processing procedure, candidate objects that do not contribute to the result are identified and pruned based on LD-distances that use as few dimensional computations as possible. DForest [53] leverages the orthogonal matrix provided by PCA [78] to reduce the dimensionality of each data object to the minimum that satisfies a fixed error during index construction. During the query processing procedure, distances are first computed in the reduced-dimensional space, and only data objects that do not satisfy prune criteria are retained for further computations. Notably, their transformation process solely rotates the data objects without altering pairwise distances.

5.2 Learing-based AKNN Search Optimization.

Previous studies have explored various methods that leverage machine/deep learning techniques to reduce the costs of various operations in AKNN search [25, 54]. Li et al. [52] leverages a gradient boosting decision tree model to predict the parameter configuration for the given query object with desired recall. LEQAT [79] models the relationship between query parameters and recall from historical data and reduces the parameter selection problem to a multiple-choice knapsack problem [22], which it then solves to determine the optimal configuration. LAN [65] adopts graph neural networks (GNN) to select an effective query entry point and to prune redundant path exploration in beam search. Baranchuk et al. [7] leverages the single-source shortest path from the query object to its K nearest neighbor objects as the oracle to guide neural network learn the routing in proximity graph search. Specifically, two separate encoders are trained through imitation learning [68], which encode query and data objects to perform similarity based decision-making during the routing process. LIMS [71] adopts the K center algorithm [36] to partition the dataset and designs an optimized disk layout based on high-quality pivots within each cluster. A recursive linear regression model predicts data locations during the query process procedure to reduce disk I/Os. Li et al. [55] designs loss functions that reduce random I/O for specific query workloads to train linear and non-linear networks to reduce random accesses. Our study is orthogonal to these studies and can be integrated with any of them to improve search performance.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose the LIDAR training pipeline and the query integration method, a novel learning-based framework to accelerate the AKNN search by deriving a learned representation tailored for this task. We propose a contrastive training pipeline to learn the non-linear mapping from the original high-dimensional representation to the AKNN-oriented representation. Building upon the learned representation, we further analyze the existing AKNN query processing produces and devise general techniques to accelerate these query processing procedures. We use two mainstream indexes as examples to demonstrate how to apply our proposed technique. Extensive experimental results show that LIDAR achieves up to 7.49× speedup on average for end-to-end performance and that each component contributes positively to overall performance. In the future, we plan to (i) further reduce the distance computation cost and (ii) explore learning-based techniques to reduce the costs of other operations in the AKNN search.

ACKNOWLEDGMENTS

This work was supported by the [...] Research Fund of [...] (Number [...]). Additional funding was provided by [...] and [...]. We also thank [...] for contributing [...].

REFERENCES

[1] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2016. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *42nd International Conference on Very Large Data Bases (VLDB)*, Vol. 9. 12.

[2] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)* 45, 6 (1998), 891–923.

[3] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems (IS)* 87 (2020), 101374.

[4] Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 931–938.

[5] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)* 37, 6 (2014), 1247–1260.

[6] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the inverted indices for billion-scale approximate nearest neighbors. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 202–216.

[7] Dmitry Baranchuk, Dmitry Persiyanov, Anton Sinitsin, and Artem Babenko. 2019. Learning to route in similarity graphs. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 475–484.

[8] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM (CACM)* 18, 9 (1975), 509–517.

[9] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. 97–104.

[10] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in Neural Information Processing Systems (NeurIPS)* 26 (2013).

[11] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France (COMPSTAT), August 22-27, 2010 Keynote, Invited and Contributed Papers*. Springer, 177–186.

[12] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems (NeurIPS)* 6 (1993).

[13] Jianlong Chang, Lingfeng Wang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2017. Deep adaptive image clustering. In *Proceedings of the IEEE international conference on computer vision (ICCV)*. 5879–5887.

[14] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 38. 17754–17762.

[15] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search*. https://github.com/Microsoft/SPTAG

[16] 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 neighborhood search. *Advances in Neural Information Processing Systems (NeurIPS)* 34 (2021), 5199–5212.

[17] Xinlei Chen and Kaiming He. 2021. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. 15750–15758.

[18] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An E cient access method for similarity search in metric spaces. In *Proceedings of the 23rd Very Large Data Bases conference, Athens, Greece (VLDB)*. Citeseer, 426–435.

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

[20] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web (WWW)* (Hyderabad, India). Association for Computing Machinery, New York, NY, USA, 577–586.

[21] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[22] Krzysztof Dudziński and Stanisław Walukiewicz. 1987. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research* 28, 1 (1987), 3–21.

[23] Karima Echihabi, Theophanis Tsandilas, Anna Gogolou, Anastasia Bezerianos, and Themis Palpanas. 2023. ProS: data series progressive k-NN similarity search and classification with probabilistic quality guarantees. *The Very Large Data Bases journal (VLDBJ)* 32, 4 (2023), 763–789.

[24] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. High-dimensional similarity search for scalable data science. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2369–2372.

[25] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. New trends in high-d vector similarity search: al-driven, progressive, and distributed. *Proceedings of the Very Large Data Bases Endowment (VLDB)* 14, 12 (2021), 3198–3201.

[26] Ronald Aylmer Fisher and Ronald A Fisher. 1971. *The design of experiments*. Springer.

[27] 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 (TPAMI)* 44, 8 (2021), 4139–4150.

[28] Yujian Fu, Cheng Chen, Xiaohui Chen, Weng-Fai Wong, and Bingsheng He. 2024. Optimizing the Number of Clusters for Billion-Scale Quantization-Based Nearest Neighbor Search. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 36, 11 (2024), 6786–6800.

[29] Jianyang Gao and Cheng Long. 2023. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 2 (2023), 1–27.

[30] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 2946–2953.

[31] Sixue Gong, Vishnu Naresh Boddeti, and Anil K Jain. 2019. On the intrinsic dimensionality of image representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 3987–3996.

[32] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2012. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)* 35, 12 (2012), 2916–2929.

[33] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. 2021. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning* 110 (2021), 393–416.

[34] 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 (SIGMOD)*. 47–57.

[35] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.

[36] Dorit S Hochbaum and David B Shmoys. 1985. A best possible heuristic for the k-center problem. *Mathematics of operations research* 10, 2 (1985), 180–184.

[37] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[38] Michael E Houle. 2017. Local intrinsic dimensionality I: an extreme-value-theoretic foundation for similarity applications. In *Similarity Search and Applications: 10th International Conference, SISAP 2017, Munich, Germany, October 4-6, 2017, Proceedings 10*. Springer, 64–79.

[39] Michael E Houle. 2017. Local intrinsic dimensionality II: multivariate analysis and distributional support. In *Similarity Search and Applications: 10th International Conference, SISAP 2017, Munich, Germany, October 4-6, 2017, Proceedings 10*. Springer, 80–95.

[40] Houdong Hu, Yan Wang, Linjun Yang, Pavel Komlev, Li Huang, Xi Chen, Jiapei Huang, Ye Wu, Meenaz Merchant, and Arun Sacheti. 2018. Web-scale responsive visual search at bing. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (KDD)*. 359–367.

[41] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 2553–2561.

[42] 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 Very Large Data Bases Endowment (VLDB)* 9, 1 (2015), 1–12.

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

[44] 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 (NeurIPS)* 32 (2019).

[45] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)* 33, 1 (2010), 117–128.

[46] Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, and Xiaofei He. 2014. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics* 44, 11 (2014), 2167–2177.

[47] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data (TBD)* 7, 3 (2019), 535–547.

[48] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track*

Proceedings. OpenReview.net.

[49] Konstantinos Lampropoulos, Fatemeh Zardbani, Nikos Mamoulis, and Panagiotis Karras. 2023. Adaptive indexing in high-dimensional metric spaces. *Proceedings of the Very Large Data Bases Endowment (VLDB)* 16, 10 (2023), 2525–2537.

[50] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems (NeurIPS)* 33 (2020), 9459–9474.

[51] Conglong Li, David G Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. 2018. Better caching in search advertising systems with rapid refresh predictions. In *Proceedings of the 2018 World Wide Web Conference.* 1875–1884.

[52] 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 (SIGMOD).* 2539–2554.

[53] Lingli Li, Wenjing Sun, and Baohua Wu. 2024. Dforest: A minimal dimensionality-aware indexing for high-dimensional exact similarity search. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2024).

[54] 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 (TKDE)* 35, 9 (2022), 8997–9018.

[55] Mingjie Li, Ying Zhang, Yifang Sun, Wei Wang, Ivor W Tsang, and Xuemin Lin. 2020. I/O efficient approximate nearest neighbour search based on learned functions. In *2020 IEEE 36th International Conference on Data Engineering (ICDE).* IEEE, 289–300.

[56] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 32, 8 (2019), 1475–1488.

[57] Yingfan Liu, Jiangtao Cui, Zi Huang, Hui Li, and Heng Tao Shen. 2014. SK-LSH: an efficient index structure for approximate nearest neighbor search. *Proceedings of the Very Large Data Bases Endowment (VLDB)* 7, 9 (2014), 745–756.

[58] 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 (TPAMI)* 42, 4 (2018), 824–836.

[59] Magdalen Dobson Manohar, Zheqi Shen, Guy Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. 2024. Parlayann: Scalable and deterministic parallel graph-based approximate nearest neighbor search algorithms. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP).* 270–285.

[60] Julieta Martinez, Joris Clement, Holger H Hoos, and James J Little. 2016. Revisiting additive quantization. In *European Conference on Computer Vision, Amsterdam (ECCV), The Netherlands, October 11-14, 2016, Proceedings, Part II 14.* Springer, 137–153.

[61] Javier Vargas Munoz, Marcos A Gonçalves, Zanoni Dias, and Ricardo da S Torres. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition (PR)* 96 (2019), 106970.

[62] nmslib. 2024. *hnswlib*. https://github.com/nmslib/hnswlib

[63] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML).* Pmlr, 1310–1318.

[64] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems (NeurIPS)* 32 (2019).

[65] Yun Peng, Byron Choi, Tsz Nam Chan, and Jianliang Xu. 2022. Lan: Learning-based approximate k-nearest neighbor search in graph databases. In *2022 IEEE 38th international conference on data engineering (ICDE).* IEEE, 2508–2521.

[66] Parikshit Ram and Kaushik Sinha. 2019. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th acm SIGKDD international conference on knowledge discovery & data mining (KDD).* 1378–1388.

[67] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, (EMNLP-IJCNLP) 2019, Hong Kong, China, November 3-7, 2019.* Association for Computational Linguistics, 3980–3990.

[68] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 40th international conference on artificial intelligence and statistics (AISTATS).* JMLR Workshop and Conference Proceedings, 627–635.

[69] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proceedings of the Very Large Data Bases Endowment (VLDB)* (2014).

[70] Joshua B Tenenbaum, Vin de Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

[71] Yao Tian, Tingyun Yan, Xi Zhao, Kai Huang, and Xiaofang Zhou. 2022. A learned index for exact similarity search in metric spaces. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 35, 8 (2022), 7624–7638.

[72] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2022. DB-LSH: Locality-Sensitive Hashing with Query-based Dynamic Bucketing. In *Proceedings-International Conference on Data Engineering (ICDE)*, Vol. 2022. IEEE, 2250–2262.

[73] Caetano Traina Jr, Roberto F Santos Filho, Agma JM Traina, Marcos R Vieira, and Christos Faloutsos. 2007. The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *The Very Large Data Bases Journal (VLDBJ)* 16, 4 (2007), 483–505.

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

[75] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An i/o-efficient disk-resident graph index framework for high-dimensional vector similarity search on data segment. *Proceedings of the ACM on Management of Data (SIGMOD)* 2, 1 (2024), 1–27.

[76] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. *Proceedings of the Very Large Data Bases Endowment (VLDB)* 13, 12 (2020), 3152–3165.

[77] Frank Wilcoxon. 1992. Individual comparisons by ranking methods. In *Breakthroughs in statistics: Methodology and distribution.* Springer, 196–202.

[78] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

[79] 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 Very Large Data Bases Journal (VLDBJ)* 32, 3 (2023), 623–645.

[80] 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 Very Large Data Bases Endowment (VLDB)* 16, 8 (2023), 1979–1991.

[81] Bolong Zheng, Zhao Xi, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S Jensen. 2020. PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search. *Proceedings of the Very Large Data Bases Endowment (VLDB)* 13, 5 (2020), 643–655.

A PROOF OF THEOREM 1

Proof. We first proof the single layer version, i.e., $f(x) = ReLU(W^\top \cdot x)$ Let's ignore the activation function ReLU for now, we prove $\|W^\top \cdot o,\ W^\top \cdot q\| \leq \|W\| \cdot \|o,\ q\|$. We define $c = o - q$ and $\hat{c} = W^\top o - W^\top q = W^\top \cdot c$. Clearly, $\|o,\ q\| = c^\top \cdot c$ and $\|W^\top \cdot o,\ W^\top \cdot q\| = \hat{c}^\top \cdot \hat{c}$. The Singular Value Decomposition (SVD) of $W \in \mathbb{R}^{d \times d'}$ is given by:

$$W = U \Sigma V^\top \tag{6}$$

where $U \in \mathbb{R}^{d \times d}$ and $V \in \mathbb{R}^{d' \times d'}$ are orthogonal matrix ($U^\top \cdot U = I_d$, $V^\top \cdot V = I_{d'}$), $\Sigma \in \mathbb{R}^{d \times d'}$ is a diagonal matrix containing the singular values.

Substitute Eq. (6) into $\|W^\top \cdot o,\ W^\top \cdot q\|$ to obtain:

$$\begin{aligned} \hat{c}^\top \cdot \hat{c} &= ((U \Sigma V^\top)^\top c)^\top \cdot (U \Sigma V^\top)^\top c \\ &= c^\top U \Sigma V^\top V \Sigma^\top U^\top c \\ &= c^\top U \Sigma \Sigma^\top U^\top c. \end{aligned} \tag{7}$$

We define $z = U^\top c$. Since U is orthogonal matrix, it preserves the Euclidean norm:

$$\|z\| = z^\top z = c^\top U U^\top c = c^\top c. \tag{8}$$

Substitute z into Eq. (8), we to obtain:

$$\hat{c}^\top \cdot \hat{c} = z^\top \Sigma^\top \Sigma z. \tag{9}$$

Since $\Sigma^\top\Sigma = diag(\sigma_1^2,\ \sigma_2^2, \ldots,\ \sigma_{d'}^2,\ 0, \ldots,\ 0)$:

$$\begin{aligned} \|W^\top \cdot o,\ W^\top \cdot q\| &= \sqrt{z^\top \Sigma^\top \Sigma z} \\ &= \sqrt{\textstyle\sum_{i=1}^{d'} \sigma_i^2 \cdot z[i]^2}. \end{aligned} \tag{10}$$

Since the Spectral norm of W is equal the $\max_i(\sigma_i)$, thus:

$$\|W^\top \cdot o,\ W^\top \cdot q\| \le \|W\| \cdot \|o,\ q\|. \tag{11}$$

Now let's consider the· activation functions $ReLU(x) = \max(0,\ x)$, we prove $\|f(o),\ f(q)\| \le \|W^\top \cdot o,\ W^\top \cdot q\|$. We define $\hat{o} = W^\top o$ and $\hat{q} = W^\top q$. Since the Euclidean distance is calculated independently for each dimension and then accumulated, we only need to prove that $(ReLU(\hat{o}[i]) - ReLU(\hat{q}[i]))^2 \le (\hat{o}[i] - \hat{q}[i])^2$ where $1 \le i \le d'$. We consider three cases:

- Case 1: Both $\hat{o}[i]$ and $\hat{q}[i]$ are positive. Since the value of a positive number after passing through the $ReLU$ function is itself, we obtain that $(ReLU(\hat{o}[i]) - ReLU(\hat{q}[i]))^2 = (\hat{o}[i] - \hat{q}[i])^2$;
- Case 2: Both $\hat{o}[i]$ and $\hat{q}[i]$ are negative. Since the value of a negative number after passing through the $ReLU$ function is 0, we obtain that $ReLU(\hat{o}[i]) - ReLU(\hat{q}[i]) = 0$. Since $(\hat{o}[i] - \hat{q}[i])^2 \ge 0$, we obtain that $(ReLU(\hat{o}[i]) - ReLU(\hat{q}[i]))^2 \le (\hat{o}[i] - \hat{q}[i])^2$;
- Case 3: $\hat{o}[i]$ is positive and $\hat{q}[i]$ is negative. We obtain that $ReLU(\hat{o}[i]) - ReLU(\hat{q}[i]) = \hat{o}[i]$. Since $\hat{q}[i]$ is negative, we obtain that $\hat{o}[i] - \hat{q}[i] \ge \hat{o}[i]$. Combining the above results, we obtain that $(ReLU(\hat{o}[i]) - ReLU(\hat{q}[i]))^2 < (\hat{o}[i] - \hat{q}[i])^2$. The same applies when the $\hat{o}[i]$ is negative and $\hat{q}[i]$ is positive. Thus, $(ReLU(\hat{o}[i]) - ReLU(\hat{q}[i]))^2 \le (\hat{o}[i] - \hat{q}[i])^2$ is always true. Through the above analysis, we obtain:

$$\|f(o),\ f(q)\| \le \|W^\top \cdot o,\ W^\top \cdot q\|. \tag{12}$$

Combining Eq. (11) and Eq. (12) imply:

$$\|f(o),\ f(q)\| \le \|W\| \cdot \|o,\ q\|. \tag{13}$$

This property can obviously be extended to cases with more layers, and the *Lipschitz constant* of multi-layer is the product of the *Lipschitz constant* of each layer. □

B EXTENDED INSIGHT 2

Fig.12 presents the distance cumulative distribution function (CDF) between objects across four real-world datasets. From Fig.12 (a)-(d), we can observe that: when $K \le 1000$, fewer than 5% objects have small distances, at least 95% of the data objects can be pruned as long as their estimated distances exceed a specific value. Fig.13 illustrates the pairwise distance distribution between data objects from two real-world datasets. It can be observed that fewer than 1% of the distances between the proximal objects exceed the overall mean. Thus, the LD-distance for distant data objects can tolerate a lower error than that for proximal objects in AKNN search.

C THE TRAINING DETAIL

In each epoch, we leverage the mini-batch stochastic gradient descent to update the model parameters. Specifically, the gradient is estimated with a random sample subset $\mathcal{B}$ of the training set, with a fixed size bs. Algorithm 5 illustrates the torch style pseudo-code for the training procedure For each mini-batch, we randomly select bs objects from the current training set as the object set of batch $\mathcal{B}$ until the training set is empty (line 1-3). We aggregate the local

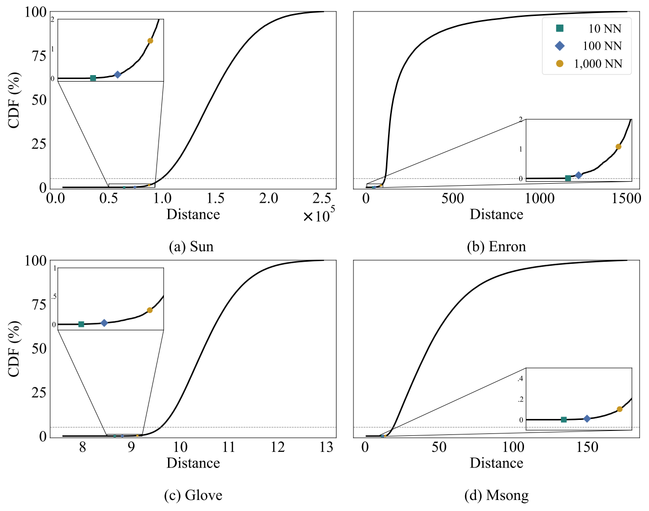

Figure 12: Illustration of the distance cumulative distribution function between query objects and data objects across four dataset. ■, ♦, and ● represent 10, 100 and 1, 000 NN, respectively. The gray dashed line represents 5%.

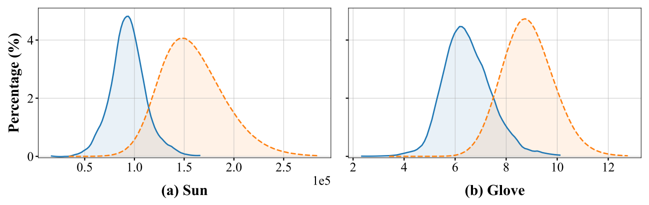

Figure 13: The Distribution of Pairwise Distance in *Sun* and *Glove*. The blue solid line represents the distance distribution between the data objects and their proximal objects (100NN), and the orange dashed line represents the distance distribution between the global data objects.

contrastive object set $\mathcal{P}(o)$ of each object $o \in \mathcal{B}$ into a set $\mathcal{A}_\mathcal{B}$ as the complete contrastive object set for this batch (line 4-6). Since the price of GPU memory is expensive, we store $\mathcal{D}'$ in DRAM or disk and only the objects in $\mathcal{B}$ or $\mathcal{A}_\mathcal{B}$ are dynamically loaded into GPU memory (line 7). We compute the loss of two arbitrary objects between $\mathcal{B}$ and $\mathcal{A}_\mathcal{B}$ by broadcast mechanism (line 9-14). Finally, we compute the gradients and update the model parameter (line 15-16).

Although constructing this training set via exhaustive search takes several hours, it is a one-time cost as long as the underlying data distribution remains relatively stable. And this time cost can be easily amortized by a large number of query requests. For example, there are hundreds of millions of online search queries on Bing per day [51].

Space Complexity Analysis. Since model training is executed using GPU-based parallel computation, and GPU architectures may vary, we only focus on the space complexity of the training process. The GPU memory usage can be divided into two components: data storage and intermediate computation. For data storage, we aggregate all $\mathcal{P}(o)$ of $o \in \mathcal{B}$ into a shared contrastive set. Compared to

Algorithm 5: Model Training

```
Input: Data Set $\mathcal{D}' = \{o_1, o_2, \dots, o_{n'}\}$, Train Set
       $\mathcal{T} = \{\mathcal{P}(o_1), \mathcal{P}(o_2) \dots, \mathcal{P}(o_{n'})\}$, Batch Size $bs$,
       Control Coefficient $\lambda$
Output: The parameter of the mapping function $W$
1  while $\mathcal{T} \neq \emptyset$ do
2      Randomly extract a mini-batch set
        $\mathcal{B} = \{o_{s_1}, o_{s_2}, \dots, o_{s_{bs}}\}$ from $\mathcal{T}$
3      $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{B}$
4      $\mathcal{A}_{\mathcal{B}} \leftarrow \emptyset$
5      for $o$ in $\mathcal{B}$ do
6          $\mathcal{A}_{\mathcal{B}} \leftarrow \mathcal{A}_{\mathcal{B}} \cup \mathcal{P}(o)$
7      Load $\mathcal{D}[\mathcal{B}]$, $\mathcal{D}[\mathcal{A}_{\mathcal{B}}]$ to HBM as tensor $\mathcal{DB}$, $\mathcal{DA}$
8                                     ▷ shape:$(bs,\ d)$, $(bs \cdot k,\ d)$
9      $\mathcal{DB} \leftarrow \mathcal{DB}$.unsqueeze(1)          ▷ shape:$(bs,\ 1,\ d)$
10     $\mathcal{DA} \leftarrow \mathcal{DA}$.unsqueeze(0)          ▷ shape: $(1,\ bs \cdot k,\ d)$
11     $d_{in} \leftarrow norm(\mathcal{DB} - \mathcal{DA}, p = 2, dim = 2)$
12                                     ▷ shape:$(bs,\ bs \cdot k)$
13     $d_{out} \leftarrow norm(f(\mathcal{DB}) - f(\mathcal{DA}), p = 2, dim = 2)$
14                                     ▷ shape:$(bs,\ bs \cdot k)$
15     $loss \leftarrow \lambda \cdot (d_{out} - d_{in})^2 + (1 - \lambda) \cdot log^2(\frac{d_{out}}{d_{in}})$
16     $loss.back()$          ▷ Calculate the gradient for this batch
17     Update the parameter $W$
```

giving an individual contrastive set for each o, the space cost of the shared contrastive set is reduced from quadratic $O(bs^2 \cdot k \cdot d)$ to linear $O(bs \cdot k \cdot d)$. Regarding intermediate computation, maximum memory usage arises during distance computation (line 11), which involves storing pairwise differences between vectors in $\mathcal{DB}$ and $\mathcal{DA}$. This step incurs a cost of $O(bs^2 \cdot k \cdot d)$. Thus, the shared contrastive set reduces GPU memory consumption by nearly half during training.

D LIDAR-AKNN CASES

D.1 The IVF Index Case

D.1.1 ***Index Introduction.*** Inverted File Index (IVF) is the most commonly used clustering-based vector index, which divides the entire data set into multiple clusters and restricts the search to a fixed number of clusters during the query processing procedure, thus avoiding an exhaustive search. Specifically, during index construction, the dataset is divided into N_c clusters by the KMeans algorithm, where N_c denotes the number of clusters. Each data object is assigned to the cluster whose centroid is closest to it. The cluster centroid vectors and the vectors within each cluster are stored sequentially. During the query processing procedure, for a query q, the distances from q to all cluster centroids are computed sequentially and sorted in ascending order. The objects within the top-*nprobe* clusters are then selected as the candidate set C, where *nprobe* is a parameter that controls the search scope. Finally, a max-heap of size K is leveraged to record the K objects closest to q when traversing the candidate set C. The radius r is dynamically updated as the distance between q and the K-th nearest object during the search. Fig.14 provides an illustrative example of the IVF structure, and Fig.15 (a) provides an illustrative example of the IVF storage access order during querying.

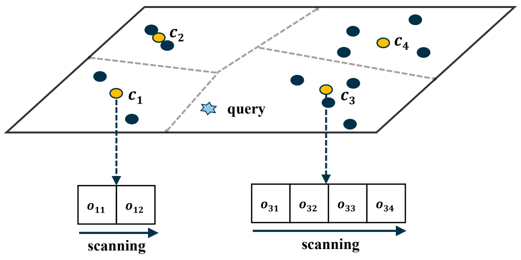

Figure 14: Example of IVF with $N_c = 4$ and $nprobe = 2$. For the query q, c_1 and c_3 are the two closest cluster centroids. Consequently, the distance between the query and all objects in clusters 1 and 3 are computed sequentially to determine the final result.

D.1.2 ***Integrate LIDAR..*** Compared to IVF, the only difference in the construction stage of IVF-LIDAR is leverage $f(\mathcal{D})$ to replace $\mathcal{D}$ for Kmeans clustering. The LIDAR and the original representation is concatenated separately while maintaining the same storage order between them, and then stored sequentially. During the query process procedure, we compute the distance between $f(q)$ and all cluster centroids to select the top-*nprobe* closest clusters as the candidate set. Subsequently, we filter the candidate set as described in Sections 3.2.2 and 3.2.3, and the rest of the query processing procedure is the same as the original IVF. Fig.15 (b) provides an illustrative example of the IVF-LIDAR storage access order during querying.

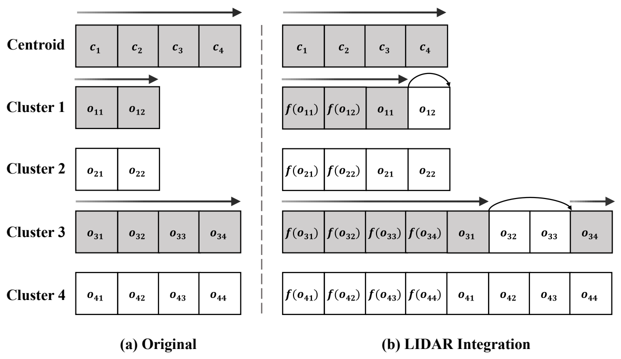
Centroid
Cluster 1
Cluster 2
Cluster 3
Cluster 4
c_1 c_2 c_3 c_4
o_{11} o_{12}
o_{21} o_{22}
o_{31} o_{32} o_{33} o_{34}
o_{41} o_{42} o_{43} o_{44}
$f(o_{11})$ $f(o_{12})$ o_{11} o_{12}
$f(o_{21})$ $f(o_{22})$ o_{21} o_{22}
$f(o_{31})$ $f(o_{32})$ $f(o_{33})$ $f(o_{34})$ o_{31} o_{32} o_{33} o_{34}
$f(o_{41})$ $f(o_{42})$ $f(o_{43})$ $f(o_{44})$ o_{41} o_{42} o_{43} o_{44}
(a) Original
(b) LIDAR Integration

Figure 15: Example of Fig.14 IVF integrate with learned representation by DDS. The gray blocks represent the data accessed during querying, and the arrows indicate the order of access.

Algorithm 6 illustrates the complete AKNN query processing procedure of IVF-LIDAR. First, we initialize the result set as an empty max-heap and set the distance threshold r to positive infinity, which is consistent with the original IVF (lines 1-2). Next, we compute the distance between $f(q)$ and all cluster centroids, selecting the top-*nprobe* closest clusters as the candidate set (lines 3–4). We then filter the candidate set based on the current distance threshold r. We store the objects and their distances that satisfy $\|f(o), f(q)\| < \alpha \cdot r$ (lines 6–9), because $\|f(o), f(q)\|$ may need to be reused when r is continuously updated. Finally, we validate the result set from the filtered candidate set (lines 10–15).

Algorithm 6: AKNN Search on IVF-LIDAR

```
Input: Clustering result $\mathbb{C}$, Query vector $q$, Number of
       neighbors $k$, Number of search cluster $nprobe$,
       Distance confidence coefficient $\alpha$.
Output: List of top-$k$ nearest neighbors.
1  $\mathcal{R} \leftarrow$ empty max-heap
2  $r \leftarrow +\infty$                         ▷ Current distance threshold
3  sort $\mathbb{C}$ based on distance between $f(q)$ and centroids
4  $C \leftarrow$ the top-$nprobe$ cluster in $\mathbb{C}$
5  for $Clu$ in $C$ do
6      $C' \leftarrow \emptyset$                  ▷ Filtered candidate set
7      for $o$ in $Clu$ do
8          if $\|f(o), f(q)\| \leq \alpha \cdot r$ then
9              $C' \leftarrow C' \cup \{\|f(o), f(q)\|, o\}$
10     for $dist$, $o$ in $C'$ do
11         if $dis \leq r$ and $\|o, q\| \leq r$ then
12             $\mathcal{R}$.push({$\|o, q\|$, $o$})
13             if $|\mathcal{R}| > k$ then
14                 $\mathcal{R}$.pop()            ▷ Remove farthest neighbor
15                 $r \leftarrow \mathcal{R}$.top().first
16 return $\mathcal{R}$
```

D.2 The HNSW Index Case

D.2.1 ***Index Introduction.*** Hierarchical Navigable Small World (HNSW) is the most commonly used PG-based index, consisting of multiple proximity graphs, where vertexes represent vectors, and each layer's vertexes form a subset of the previous layer. To this day, it remains one of the best performing PG-based indexes. During construction, the graph starts empty and data objects are inserted sequentially. Each object is assigned a random maximum level L based on an exponentially decaying probability distribution. At each level, the algorithm identifies the M objects closest to the new object o^* by its search strategy and then connects them to o^*. During querying, a greedy search begins from the highest layer's entry object, progressively moving closer to the query q until it reaches level 0. At the base layer, a beam search starts from the last encountered vertexe in the greedy search. Specifically, this process maintains two sets: the candidate set C (a min-heap, initialized with the start vertexe) and the result set $\mathcal{R}$ (a max-heap, initially empty, with a maximum size of ef), where ef controls the search scope, similar to *nprobe* in IVF. The key of the heap is the distance between the data object in the heap and q. Each time a point o is extracted from the top of C, it is added to $\mathcal{R}$, and its unvisited neighbors are inserted into C. The search terminates when the size of $\mathcal{R}$ reaches ef and the top object in C is further from q than the top object in $\mathcal{R}$. Finally, the K objects closest to q in $\mathcal{R}$ will be returned as the result set. The radius r is dynamically updated as the distance between q and the ef-th nearest object during the search. Fig.16 provides an illustrative example of the HNSW structure.

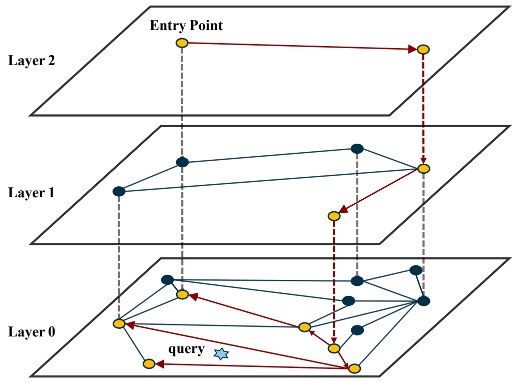

Figure 16: Example of a three-layer HNSW. The orange vertexes indicate the vertexes visited during the search, while the red edges depict the explored path.

D.2.2 ***Integrate LIDAR.*** Compared to HNSW, the only difference in the construction stage of HNSW-LIDAR is leverage $f(\mathcal{D})$ to replace $\mathcal{D}$ for graph construction and store connected LIDAR and original representation in the storage order designed by the vanilla HNSW. During the query process procedure, we start from the top layer and perform a greedy search to find the closest vertex to $f(q)$, continuing until we reach the base layer. The key difference in this step from the original HNSW is that the LD-distance alone is used as the selection criterion for the search path. Subsequently, we filter the candidate set as described in Sections 3.2.2 and 3.2.3, and the rest of the query processing procedure is the same as the original HNSW.

Algorithm 7 illustrates the complete AKNN query processing procedure of HNSW-LIDAR. First, we initialize the candidate set and the result set as an empty min-heap and max-heap, respectively, and set the distance threshold to positive infinity, which is consistent with the original HNSW (lines 1-2). Next, we start from the top layer and perform a greedy search to find the closest vertex to $f(q)$, continuing until we reach the base layer (lines 3-5). The key difference in this step from the original HNSW is that the LD-distance alone is used as the selection criterion for the search path. Finally, we perform a beam search at the base layer, as in the original HNSW (lines 6-19), but leverage the LD-distance to prune candidate objects when expending the candidate set (lines 11-12).

E ANALYSIS THE GRADIENT OF LIDAR LOSS

The gradient of $\mathcal{J}(u, v)$ w.r.t. $f(u)$ is given by:

$$\nabla_{f(u)} \mathcal{J}(u, v) = 2 \cdot (\lambda \cdot (d_{out} - d_{in}) + (1 - \lambda) \cdot \frac{log \frac{d_{out}}{d_{in}}}{d_{out}}) \cdot \frac{f(u) - f(v)}{d_{out}}. \quad (14)$$

The gradient w.r.t. $f(v)$ is the same in magnitude but in the opposite direction. As shown in Eq. (14), the LIDAR loss shares the same gradient direction as the MSE loss but produces smaller gradient magnitudes for distant object pairs.

Algorithm 7: AKNN search on HNSW-LIDAR

```
Input: HNSW Graph $\mathbb{G}$, Query object $q$, Number of
       neighbors $k$, Size of max heap $ef$, Top layer entry
       Point $e$, Distance confidence coefficient $\alpha$.
Output: List of top-$k$ nearest neighbors.
1  $C$, $\mathcal{R}$ ← empty min-heap and max-heap
2  $r \leftarrow +\infty$                          ▷ Current distance threshold
3  while the base layer in $\mathbb{G}$ is not reached do
4      $e$ ← Greedy search in the current layer to find the
         nearest object of $f(q)$
5      Move to the next layer in $\mathbb{G}$
6  push $\{\|e, q\|, e\}$ to $C$ and $\mathcal{R}$
7  while $r$ is updated in the last iteration or $|\mathcal{R}| < ef$ do
8      $u \leftarrow C$.top()
9      $C$.pop()
10     for $v$ in the unvisited neighbors of $u$ do
11         if $\|f(v), f(q)\| > \alpha \cdot r$ then
12             continue     ▷ Purning the non-contribute object
13         $C$.push($\{\|v, q\|, v\}$)
14         if $\|v, q\| < r$ then
15             $\mathcal{R}$.push($\{\|v, q\|, v\}$)
16             if $|\mathcal{R}| > ef$ then
17                 $\mathcal{R}$.pop()          ▷ Remove farthest neighbor
18                 $r \leftarrow \mathcal{R}$.top().first
19 return top-$K$ nearest neighbors in $\mathcal{R}$
```

F ADDITIONAL EXPERIMENTS

F.1 Construction Information

In this subsection, we compare our method with eight baseline methods on nine datasets. Fig.17 illustrates the index construction time and index size, respectively.

From Fig.17 (a), we observe that the construction times of IVF-LIDAR and HNSW-LIDAR decrease 67.87% and 56.09% on average, respectively, compared to the original IVF and HNSW. Moreover, IVF-LIDAR achieves the shortest construction time among clustering-based methods, while HNSW-LIDAR also achieves that among PG-base method. Although transforming $\mathcal{D}$ into its learned representation $f(\mathcal{D})$ introduces additional time cost, the decreased construction cost of building from $f(\mathcal{D})$ fully trades off this additional cost. ADSampling-IVF incurs a construction time longer than IVF because it cannot leverage LD-distances during the KMeans clustering stage and requires an additional dataset rotation step. In contrast, Adsampling-HNSW achieves shorter construction time than HNSW on some datasets, because graph construction requires AKNN search. NSSG exhibits the highest construction time among all methods, as it requires building a KNN graph and computing angles between edges to determine whether to retain or remove them. Due to the prohibitive time cost of PG-based index construction, several algorithms failed to build on certain datasets.

From Fig.17(b), we observe that the index sizes of IVF-LIDAR and HNSW-LIDAR increase 19.71% and 10.76% on average, respectively, compared to the original IVF and HNSW. Although both introduce the same amount of additional storage from the learned representation, the relative increase is smaller for HNSW-LIDAR because PG-based indexes require more space to store edges.

F.2 Impact of K

In this subsection, we investigate the impact of the number of nearest neighbors for the search by varying this number within the range of {10, 30, 50, 70, 100}. Fig.18 illustrates the query time-recall curve. It can be observed that the curves shift toward the upper-left corner as K increases, indicating that the query overhead increases. An exception occurs for HNSW on Glove with $K = 10$, where the performance is comparatively worse. This anomaly arises due to the high LID of Glove, i.e.,objects are densely clustered in close proximity, requiring a large candidate set even for small K-NN searches.

F.3 Evaluation of the randomness of storage access

In this section, we evaluate the number of random storage accesses during the query process procedure in IVF-LIDAR under DDS layout. Tab. 6 illustrates the average number of random storage accesses, the theoretical maximum number of random storage accesses, and the result of the Permutation Test. The experimental results empirically verify that the number of random accesses will not reach the theoretical maximum value. The result of the Permutation Test confirm that the improvement of the DDS layout for the two-stage AKNN search is statiscally significant at the 95% confidence level.

F.4 Compared with the existing dimensionality reduction methods

In this subsection, we evaluate the distance ratios between LD-distances and FD-distances among different dimensionality reduction methods. Fig. 19 illustrates the distance ratios (i.e., LD-distance/FD-distance) for several well-known methods, including PCA [78], PQ [45], OPQ [30], AutoEncoder [35], and IsoMap [70] with global and local range (100NN) in *Glove*. It can be observed that our method maintains the best distance ratio, and our distance ratios for proximal objects are significantly lower than those for distant objects. This aligns with our objectives, thus facilitating more effective optimization of AKNN search performance.

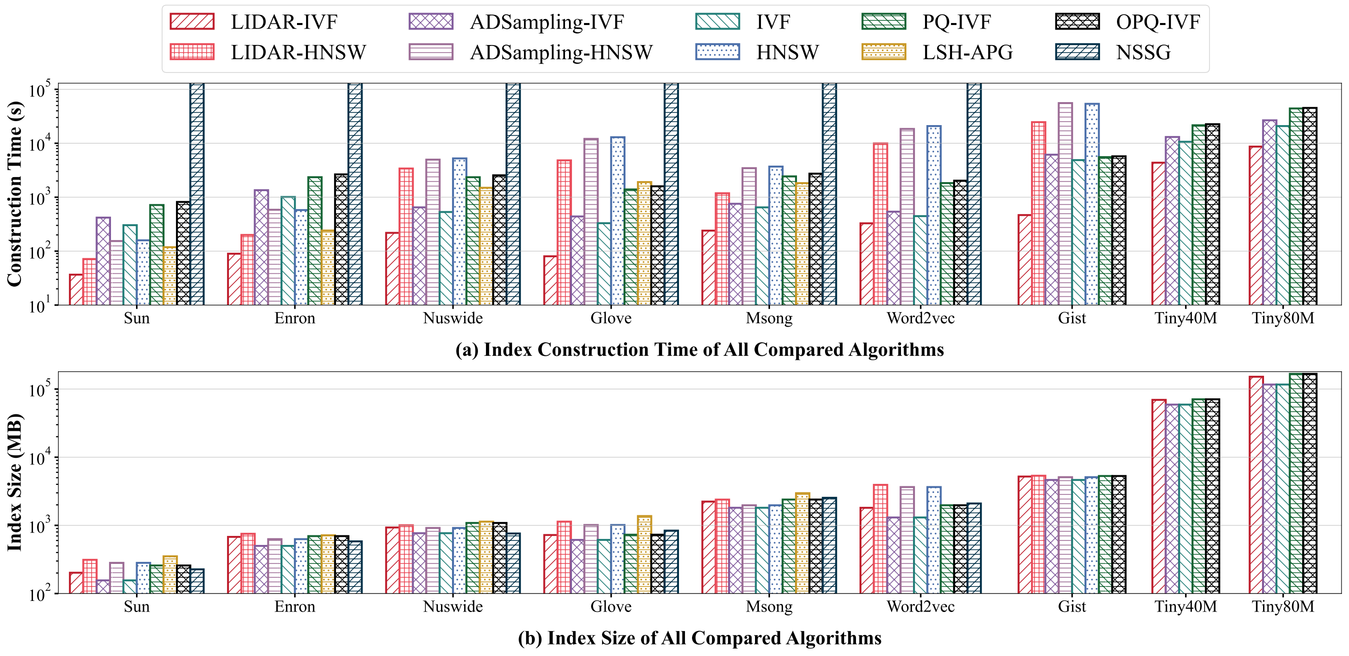

Figure 17: Construction Information of All Compared Algorithms.

Table 6: Evaluation of the number of random storage accesses.

Dataset	Sun	Enron	Nuswide	Glove	Msong	Word2vec	Gist
The number of random storage accesses	502.56	1085.45	9544.62	9586.51	1117.6	6553.02	7097.68
The theoretical maximum number of random storage accesses	761.51	1854.63	16683.11	17938	1448.54	9663.76	8840.48
p-value	45.22	53.29	76.68	80.04	77.43	80.28	75.98

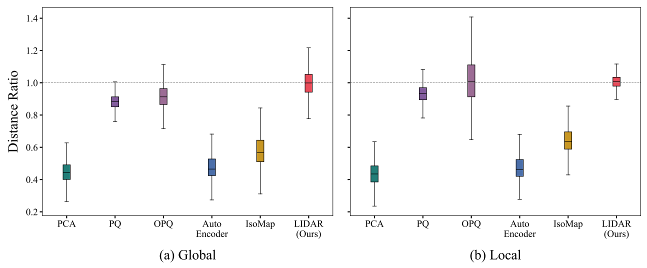

Figure 19: The distance ratio between LD-distance and FD-distance in *Glove*.

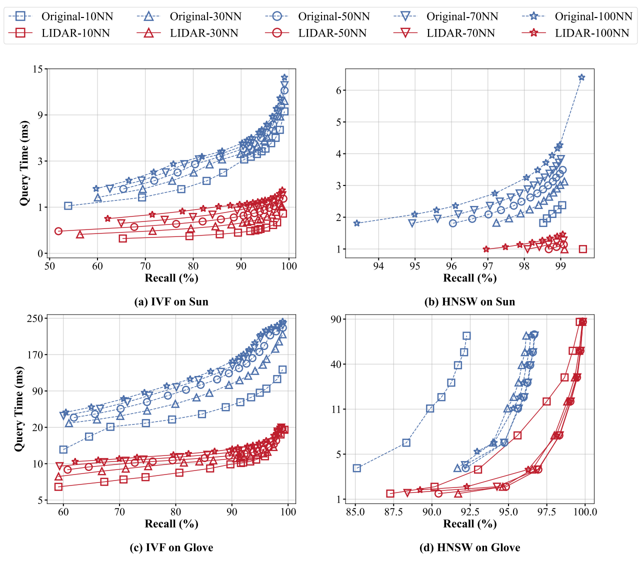

Figure 18: The QT-Recall curve with different *K*.