Learning Distance-Aware Space Partitions for Approximate Nearest Neighbor Search

Zhenyu Zhang, Junlin Shang, Kailing Li, Jiannan Li, Mengqi Tian, and Xiaoling Wang (✉)

East China Normal University, Shanghai, China
{zhenyuzhang,jlshang,71215901006,51265901065}@stu.ecnu.edu.cn
likailing2024@gmail.com, xlwang@cs.ecnu.edu.cn

Abstract. Approximate Nearest Neighbor Search (ANNS) is a fundamental problem widely applied in information retrieval and data mining, with graph-based methods gaining particular interest due to their outstanding efficiency and query accuracy. However, as the scale of high-dimensional data continues to grow, efficient space partitioning becomes essential for supporting distributed vector data processing. We model space partitioning as a balanced graph partitioning problem, but traditional graph partitioning algorithms, which primarily aim to minimize edge cuts, may inadvertently separate closer neighbors. To overcome this limitation, we propose a distance-aware multilevel partitioning algorithm that combines distance-aware coarsening and refinement strategies, effectively reducing the incidence of closer nodes being cut. We train a lightweight model to predict partition where the neighbors of a query point are most likely to be located by minimizing the Kullback-Leibler (KL) divergence between the predicted partition distribution and the ground truth distribution derived from neighbor counts. Additionally, our approach dynamically repartitions boundary nodes after each training iteration, enabling alternating updates that optimize both the classifier and the quality of the partitioning. Experimental results demonstrate the superiority of the proposed method to the baselines in the tradeoff among search efficiency and partitioning quality.

Keywords: Approximate nearest neighbor search · Graph partitioning · Learning to index.

1 Introduction

The approximate nearest neighbor search (ANNS) in a high dimensional space is of great importance in applications such as databases, information retrieval, data mining, pattern recognition, and machine learning [24]. ANNS represents objects as points in $\mathbb{R}^d$ space and, given a dataset $\mathcal{D} \subseteq \mathbb{R}^d$ and a query $q \in \mathbb{R}^d$, aims to return the k closest vectors in $\mathcal{D}$ to q based on a chosen distance measure, such as Euclidean distance. Currently, many approximate nearest neighbor methods

Z. Zhang and J. Shang - Equal contribution.

have been proposed, which are mainly divided into four categories: tree-based methods, quantization-based methods, graph-based methods, and hash-based methods. Although graph-based methods offer the best trade-off between recall and throughput, the continuous growth of high-dimensional data makes it increasingly challenging for single-machine solutions to meet the storage demands of indexing structures [14]. Therefore, efficient space partitioning is crucial for supporting distributed vector databases [6, 7].

Space partitioning methods partition the $\mathbb{R}^d$ into m partitions, including hashing-based and tree-based methods [15]. Compared to traditional indexing techniques, space partitioning presents several significant benefits. Firstly, space partitioning significantly enhances query efficiency by narrowing the search range. By dividing data into distinct space regions, queries can focus solely on relevant sub-regions, thereby reducing the number of elements that need to be examined. Secondly, partitioning is particularly well-suited for distributed computing environments, allowing different partitions to be allocated to separate machines, which supports concurrent search operations and boosts computational efficiency through independent local nearest neighbor searches [23].

It has been recently observed that Neural LSH utilizes graph partitioning algorithm to partition the constructed k-NN graph and achieves superior performance over quantization-based and tree-based partitions by employing supervised classification [6]. However, the optimization objective in graph partitioning—minimizing cut edges—does not fully align with the goals of space partitioning. Minimizing cut edges in space partitioning often neglects the distance information between points, which can result in closer neighbors being separated by cut edges, while more distant neighbors are unintentionally grouped into the same partition. For instance, in traditional graph partitioning algorithms, both Fig. 2(c) and (d) achieve the same cut edge count of two, reaching the optimization limit based solely on edge cuts. However, by observing that nodes 6 and 7 are closer neighbors to node 5 than other nodes, we see an opportunity to enhance partition quality by incorporating distance-based weights.

To overcome these challenges, we propose an optimized multilevel partitioning algorithm that integrates inter-point distances, preserving the locality of similar data points within each partition. We also introduce a lightweight model to predict the partition where the neighbors of a query point are most likely to be located, incorporating partition information from neighboring nodes as soft labels to enhance training. Additionally, after each training iteration, the model reassigns boundary nodes, further improving partition quality through joint optimization of the classifier and partitioning results. The contributions of this paper can be summarized as follows:

- We propose a multilevel partitioning algorithm that incorporates distance-aware coarsening and refinement strategies. This method effectively mitigates the issue of separating closer neighbors, leading to higher-quality partitions suited for high-dimensional data processing.
- We develop a lightweight model that predicts the partition where the neighbors of a query point are most likely to be located. By minimizing the KL

divergence between the predicted partition distribution and the ground truth distribution derived from the neighbor counts, the model assigns higher prediction probabilities to partitions with a more concentrated neighbor distribution.
- Our method features a dynamic repartitioning process that updates boundary nodes after each training iteration. This allows for alternating updates that optimize both the classifier's performance and the quality of the partitions, ensuring robust and efficient search operations.
- Experimental results demonstrate the effectiveness of our method, which achieves a superior trade-off between search efficiency and partitioning quality compared to the baselines.

2 Related Work

2.1 Space partitioning Methods

Space partitioning is essential in ANNS and can be divided into hashing-based and tree-based methods. **Hashing-Based Methods:** LSH [5] is a key approach, mapping similar points into the same "buckets" with high probability. LSH is adaptable to various distance metrics like Euclidean and cosine similarity, providing sublinear query time by limiting searches to relevant buckets, making it computationally efficient with low memory usage. **Tree-Based Methods:** Tree-based algorithms use structured trees to partition data, employing pruning strategies for efficient search. KD-trees [3], for instance, divide space into hyperplanes along feature dimensions, forming a binary tree. Balanced K-means trees [9] and Ball trees [13] use clustering and distance-based splitting to create hierarchical structures that capture data relationships.

2.2 Graph Partitioning

The research about graph partitioning can be categorized into vertex-cut strategy and edge-cut strategy. In vertex-cut strategy, the set of edges E is divided into k partitions by assigning each edge to exactly one partition $p \in P = \{p_1, \ldots, p_k\}$ with $\bigcup_{i=1}^{k} p_i = E$. In edge-cut strategy, the set of vertices V is divided into k partitions by assigning each vertex v to exactly one partition $p \in P = \{p_1, \ldots, p_k\}$ with $\bigcup_{i=1}^{k} p_i = V$. An edge $e = (u, v)$ is cut if both u and v are assigned to different partitions. The edge-cut strategy aims to minimize the number of cut edges while balancing the partition sizes in terms of the number of vertices. Classic algorithms for this task include METIS [10] and KaHIP [17], which efficiently optimize edge-cut minimization while maintaining partition balance. We define E_{cut} as the set of cut edges, whose endpoints are referred to as **Boundary nodes**. As shown in Fig. 2(a), **METIS** employs a multilevel graph partitioning algorithm that recursively coarsens the input graph, partitions the smaller graph, and then refines the partitions to minimize edge cuts while maintaining load balance. Similarly, **KaHIP** is a multilevel graph

partitioning algorithm that introduces sophisticated strategies like Flow-based and KaFFPa (Karlsruhe Fast Flow-based Partitioning) to achieve superior partitioning performance.

2.3 Learning to Index

Learning-to-index techniques use machine learning to create indexes tailored to data distributions, focusing on both index structures and models [11]. Index structures are optimized through combinatorial methods, often using greedy algorithms, while index models are learned from training data. In contrast, learning-to-partition methods apply machine learning to space partitioning, with notable methods like Learning-to-Hash (LTH) [22], BLISS [8], LIMS [20], and Neural LSH [6]. These approaches address load-balance constraints to reduce partitioning bias. LTH learns both a hashing function and hash codes, by minimizing Hamming distances for nearby vectors while maintaining load balance through combinatorial regularization. LIMS is a learned index that combines clustering, pivot-based transformations, and machine learning to efficiently process similarity queries in high-dimensional spaces. To improve hyperplane-based partitioning, Neural LSH and BLISS integrate neural networks into their models. The key difference is that BLISS updates both the index structure and model, while Neural LSH uses a pre-built structure from balanced graph partitioning.

3 Our method

3.1 Overall Framework

In this paper, as shown in Fig. 1, we propose a novel method that combines a distance-aware multilevel partitioning algorithm with a Multi-Layer Perceptron (MLP) model to predict partition where the neighbors of a query point are most likely to be located.

Step 1. Initially, we construct the graph index and model the space partitioning as a balanced graph partitioning problem. To address this, we propose a multilevel partitioning algorithm that incorporates node distance information, effectively generating balanced and meaningful partitions while preserving the local structure of the data.

Step 2. After the initial graph partitioning, we implement a lightweight two-layer MLP classifier to predict the most likely partition for a query point's neighbors. This classifier outputs a probability distribution over partitions, guiding queries toward the most relevant regions and improving both accuracy and efficiency in nearest neighbor search. A key challenge arises when a query point's neighbors are dispersed across multiple partitions with varying densities, which can lead to suboptimal recall. To address this, we employ soft labels during training, assigning probabilities to partitions based on their share of the query's neighbors, thereby prioritizing denser partitions. The model is trained by minimizing the Kullback-Leibler (KL) divergence between its predicted partition distribution and the ground truth distribution derived from actual neighbor counts

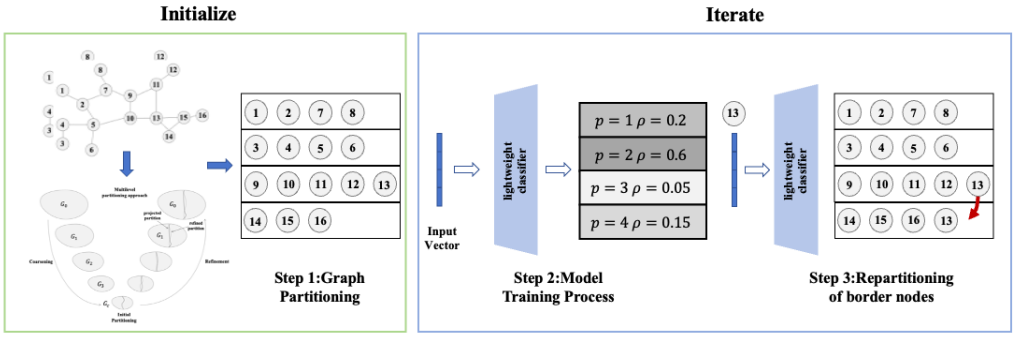

Fig. 1: In the Initialization phase, Step 1 constructs the graph index, such as NSG, and employs a distance-aware multilevel graph partitioning algorithm to create an initial partition that preserves the local structure. In the Iterate phase, Step 2 trains a lightweight classifier to predict partition memberships based on query vectors, providing probability scores for each partition. In Step 3, these predictions guide the reassignment of boundary nodes in a repartitioning step, enhancing partition balance and preserving proximity.

across partitions. This optimization ensures that the model's predictions align closely with the true neighbor distribution, enhancing recall performance and improving the retrieval of relevant neighbors.

Step 3. Since the initial partitioning may lead to imbalances, there is room for further optimization to enhance partition quality. To address this, we introduce a repartitioning step after each training iteration. In this step, the MLP model assists in reallocating boundary nodes among partitions to achieve a more balanced distribution. By iteratively refining the partitioning, this approach not only improves the balance across partitions but also enhances the overall efficiency of the nearest neighbor search.

3.2 Graph Partitioning

Given a dataset $\mathcal{D} \subseteq \mathbb{R}^d$ of n points, and a number of partitions m, our goal is to find a partition $\mathcal{P}$ of $\mathbb{R}^d$ into m partitions with the following properties:

(1) ***Balanced.*** Each partition should contain a number of data points that does not significantly exceed $(1+\gamma)\frac{n}{m}$, where γ is the balance coefficient.
(2) ***Minimizing Intra-Partition Distance.*** The partitioning should aim to minimize the distances between points within each partition. By clustering closer points together, we can reduce the need for cross-partition queries, enhancing search efficiency and query performance.
(3) ***Locality Sensitive.*** For a typical query point $q \in \mathbb{R}^d$, most of its nearest neighbors should belong to the same partition $\mathcal{P}$. We assume that both queries and data points are drawn from similar distributions.

Formally, we seek a partition $\mathcal{P}$ that minimizes the separation of nearest neighbors across different partitions. Specifically, we define the loss function as: $\mathbb{E}_q\left[\sum_{d\in N_k(q)} \mathbf{1}_{\mathcal{P}(d)\neq\mathcal{P}(q)}\right]$, where $N_k(q)$ denotes the k-nearest neighbors of q, and $\mathbf{1}_{\mathcal{P}(d)\neq\mathcal{P}(q)}$ is an indicator function equal to 1 when d and q are assigned to different partitions, and 0 otherwise. Minimizing this loss function helps to keep neighboring points within the same partition, thus preserving local structure and reducing inter-partition communication. Additionally, this partitioning is subject to a balance constraint: each partition should contain at most $(1+\gamma)\frac{n}{m}$ data points, where n is the total number of data points, m is the number of partitions, and γ is a balance parameter. This constraint ensures that partition sizes remain relatively even, avoiding imbalances that could adversely affect query performance.

We state and prove a theorem that, under certain mild assumptions, shows that the graph of a dataset $\mathcal{D} \subset \mathbb{R}^d$ can be partitioned by a hyperplane such that the resulting cut is sparse (i.e., has few crossing edges) while maintaining similar sizes for the two parts. The theorem is based on the framework of [1] and uses spectral techniques.

Theorem 1. *Let $\mathcal{D} \subset \mathbb{R}^d$ be a dataset, and let $G = (V, E)$ represent the graph of $\mathcal{D}$, where each node $v \in V$ corresponds to a point in $\mathcal{D}$, and each edge $(v, w) \in E$ connects v to one of its k-nearest neighbors. Assuming that the typical distance between nearest neighbors is significantly smaller than the distance between randomly chosen points, there exists a hyperplane $H = \{x \in \mathbb{R}^d \mid \langle a, x\rangle = b\}$ such that: The dataset $\mathcal{D}$ is divided into two subsets, $\mathcal{D}_1 = \{d \in \mathcal{D} \mid \langle a, d\rangle \leq b\}$ and $\mathcal{D}_2 = \{d \in \mathcal{D} \mid \langle a, d\rangle > b\}$, with $|\mathcal{D}_1| \approx |\mathcal{D}_2|$, ensuring a balanced partition.*

Proof. Define the adjacency matrix A_G of G, normalized by $2kn$, so that the sum of all entries equals 1. This normalization induces a probability distribution over $\mathcal{D} \times \mathcal{D}$, which we denote by D_{close}. Define $\rho_G(p) = \sum_{p'}(A_G)_{p,p'}$ for each point $p \in \mathcal{D}$, which represents the row-sum probability for each node. Set $D_G = \text{diag}(\rho_G)$, and let $L_G = D_G - A_G$ denote the Laplacian matrix of G.

Define α as the typical squared distance between nearest neighbors and β as the typical squared distance between randomly chosen points. Since $\alpha \ll \beta$ by assumption, the distance ratio $\frac{\alpha}{\beta}$ should be small. To utilize this property, we consider all possible projections of points in $\mathcal{D}$ onto individual coordinates and find an optimal projection coordinate i^* such that:

$$\frac{\sum_{p,p'\in\mathcal{D}}(A_G)_{p,p'} \cdot (p_{i^*} - p'_{i^*})^2}{\sum_{p,p'\in\mathcal{D}} \rho_G(p)\rho_G(p') \cdot (p_{i^*} - p'_{i^*})^2} \leq \frac{\alpha}{\beta} \tag{1}$$

Let $y \in \mathbb{R}^{\mathcal{D}}$ be the vector corresponding to the i^*-th coordinate of each point.

The expression $y^T L_G y$ represents the numerator, as it corresponds to distances between connected nodes in the graph. For the denominator, consider a complete graph H on $\mathcal{D}$, with edge weights $\rho_G(p)\rho_G(p')$. Define the adjacency matrix $A_H = \rho_G\rho_G^T - D_G^2$ and the Laplacian $L_H = D_H - A_H = D_G - \rho_G\rho_G^T$. Then:

$$\frac{y^T L_G y}{y^T(D_G - \rho_G\rho_G^T)y} \leq \frac{\alpha}{\beta} \tag{2}$$

To ensure no directional bias, define $\tilde{y} = y - c \cdot 1$, where $c = \frac{y^T \rho_G}{1^T \rho_G}$, ensuring $\tilde{y} \perp \rho_G$. This adjustment keeps $L_G \tilde{y} = L_G y$ and $L_H \tilde{y} = L_H y$, leading to:

$$\frac{\tilde{y}^T L_G \tilde{y}}{\tilde{y}^T D_G \tilde{y}} = \frac{y^T L_G y}{y^T (D_G - \rho_G \rho_G^T) y} \leq \frac{\alpha}{\beta} \tag{3}$$

Using Cheeger's inequality [4], we conclude that there exists a threshold y_0 such that:

$$\frac{\sum_{p_1,p_2:\tilde{y}_{p_1} \leq y_0, \tilde{y}_{p_2} > y_0} (A_G)_{p_1,p_2}}{\min\left\{\sum_{p:\tilde{y}_p \leq y_0} \rho_G(p), \sum_{p:\tilde{y}_p > y_0} \rho_G(p)\right\}} \leq \sqrt{\frac{\tilde{y}^T L_G \tilde{y}}{\tilde{y}^T D_G \tilde{y}}} \leq \sqrt{\frac{2\alpha}{\beta}} \tag{4}$$

This shows that there exists a hyperplane H that divides $\mathcal{D}$ into two balanced subsets $\mathcal{D}_1$ and $\mathcal{D}_2$ such that the number of edges crossing the partition is minimized. This achieves a sparse cut, ensuring that most nearest neighbors remain in the same partition, preserving local connectivity and supporting efficient query processing across partitions.

Traditional graph partitioning algorithms mainly focus on minimizing cut edges and balancing partition sizes, often overlooking the distances between neighbors that are split. This oversight can result in closer neighbors—which are crucial for preserving local structure and query efficiency—being assigned to different partitions. To better preserve local neighborhood structure, we propose incorporating distance-based edge weights. This approach prioritizes maintaining connections between nearby nodes, which are essential for preserving local structure and enhancing query efficiency. Below, we outline the steps involved in implementing this distance-aware partitioning.

Selecting an Appropriate Distance Metric. Depending on the nature of the dataset, various distance metrics can measure the proximity between nodes. Common choices include Euclidean Distance, Cosine Similarity, and Manhattan Distance. Euclidean Distance, calculated as $d_{uv} = \|\vec{u} - \vec{v}\|_2 = \sqrt{\sum_{i=1}^{d} (u_i - v_i)^2}$, is suitable for continuous numerical features. Cosine Similarity, defined as $d_{uv} = 1 - \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|}$, is ideal for high-dimensional vector data, such as text embeddings. Manhattan Distance, calculated as $d_{uv} = \sum_{i=1}^{d} |u_i - v_i|$, works well in grid-like or discrete spaces. Based on the selected distance metric, each edge in the graph can be assigned a weight, thus helping to prioritize connections between closer nodes during partitioning.

Calculating Distance-Based Edge Weights. After selecting the distance metric, we assign weights to each edge inversely proportional to the distance between its nodes. This ensures that edges connecting closely related nodes receive higher weights. This weighting is calculated using the formula $w_{uv} = \frac{1}{d_{uv} + \epsilon}$, where d_{uv} represents the distance between nodes u and v, and ϵ is a small constant to avoid division by zero. This approach ensures that edges between closer nodes are prioritized in the partitioning process, thereby preserving local connectivity more effectively.

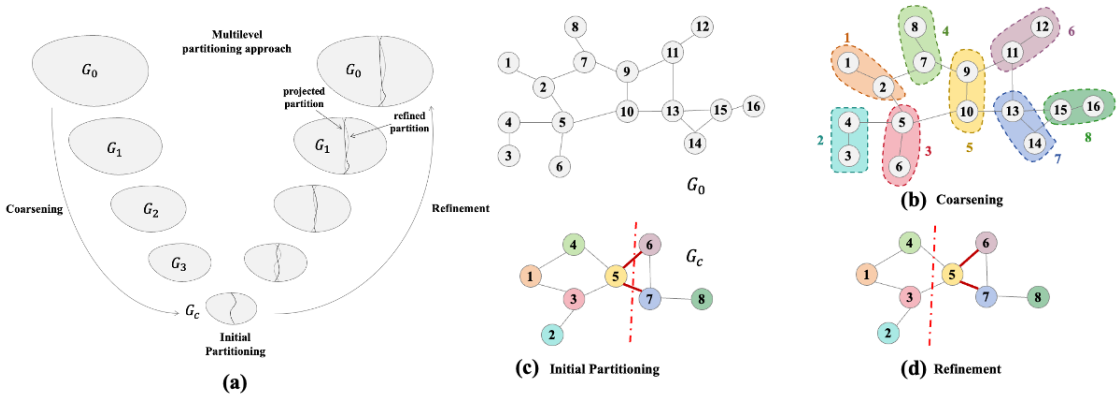

Fig. 2: Distance-Aware Multilevel Graph Partitioning: Coarsening, Initial Partitioning, and Refinement.

Modifying the Optimization Objective to Minimize Weighted Cuts. Traditional graph partitioning algorithms minimize the total number of cut edges without considering the strength or significance of each edge. By incorporating distance-based weights, we adjust the optimization objective to minimize the sum of the weights of cut edges, rather than merely the count. The modified objective becomes:

$$\text{Minimize} \sum_{(u,v)\in E_{\text{cut}}} w_{uv} \tag{5}$$

where E_{cut} denotes the set of edges cut by the partitioning, and w_{uv} is the weight of each edge, reflecting the proximity between connected nodes. This objective effectively reduces the likelihood of separating closely connected nodes, thereby enhancing the quality of the partitioning with respect to preserving local structure. To further improve partition quality, we propose a distance-based multilevel partitioning approach, consisting of three primary phases: coarsening, initial partitioning, and refinement. This approach leverages distance weights at each stage to enhance the preservation of local neighborhood structures, as illustrated in Fig. 2.

Coarsening Phase. As shown in Fig. 2(a) and (b), the coarsening phase progressively simplifies the graph by merging pairs of nodes, with priority given to nodes connected by edges with higher distance-based weights, representing closer node pairs. By consolidating nodes with their nearest neighbors, this phase preserves critical neighborhood relationships, maintaining the graph's local structure, which is essential for accurate partitioning. This weighted coarsening approach allows the graph to retain its proximity-based structure, even as the graph size is reduced.

Initial Partitioning. After the coarsening phase, as illustrated in Fig. 2(c), the smallest, most simplified version of the graph undergoes an initial partitioning using traditional partitioning techniques. Although distance weights are not

directly applied here, the prior coarsening has already structured the graph to align closely related nodes within the same regions, laying a strong foundation for high-quality partitions that preserve local connectivity.

Refinement Phase. Finally, in the refinement phase, shown in Fig. 2(d), the initial partitions are projected back onto the original graph in an iterative manner. Here, distance-based weights are critical in adjusting partition boundaries to minimize the number of cut edges with high weights, effectively reducing the chances of separating closely related nodes. This refinement process moves nodes across partitions as necessary, optimizing the objective of minimizing weighted cuts, enhancing connectivity within each partition, and maintaining the integrity of the graph's local structure.

3.3 Model Structure and Training Process

We employ a lightweight MLP model to predict the potential partitions where the neighbors of a query point may reside. The model architecture consists of multiple fully connected layers, transforming the input query vector into a probability distribution over partitions. The MLP model takes the query vector q as input and generates a latent representation through L hidden layers. Each hidden layer performs a linear transformation followed by a non-linear activation function (e.g., ReLU):

$$z^{(l)} = \sigma\left(W^{(l)} z^{(l-1)} + b^{(l)}\right), \quad l = 1, 2, \ldots, L \tag{6}$$

where $z^{(0)} = q$. The output layer applies a softmax function to produce a probability distribution over all possible partitions:

$$P(y|q) = \text{softmax}\left(W_{\text{out}} z^{(L)} + b_{\text{out}}\right) \tag{7}$$

The training dataset consists of pairs of query points $\{q_i\}$ and their corresponding sets of nearest neighbors g_i. For each query point q_i, we determine the distribution of its nearest neighbors across different partitions.

The MLP model generates a probability distribution $P(y|q)$ over partitions for each query point. Since a query's neighbors may span multiple partitions, and the spread of these neighbors across partitions may vary, we use soft labels in the training process to capture this distributional variation. Specifically, the soft label probability for partition y is defined as: $P(y|g_i) = \frac{|g_i \cap y|}{|g_i|}$. To train the model, we minimize the KL divergence between the predicted partition distribution $P(y|q)$ and the target distribution $P(y|g_i)$, which reflects the actual distribution of neighbors across partitions. This objective encourages the model to approximate the true distribution of neighbors, improving the overall recall in nearest neighbor search.

To further enhance the model's learning of partition distributions, we introduce soft labels that represent the distribution of neighbors across partitions. This approach enables the model to capture not only the unique characteristics

of each query's neighbors but also the varying distributions of these neighbors across partitions. For each partition p_j of query q_i, the soft label $P_{\text{true}}(p_j|q_i)$ is defined as: $P_{\text{true}}(p_j|q_i) = \frac{m}{k}$, where m is the number of neighbors of q_i in partition p_j, k is the total number of neighbors of q_i.

To align the model's predicted distribution $P(y|q_i)$ with the actual neighbor distribution $P_{\text{true}}(p_j|q_i)$, we minimize the Kullback-Leibler (KL) divergence between the predicted and true distributions. The loss function for each query q_i is defined as:

$$L(q_i) = KL\left(P_{\text{true}}(y|q_i) \| P(y|q_i)\right) = \sum_j P_{\text{true}}(p_j|q_i) \log \frac{P_{\text{true}}(p_j|q_i)}{P(p_j|q_i)} \tag{8}$$

The model's objective is to minimize the average KL divergence over all queries, defined as:

$$L_{\text{avg}} = \frac{1}{N} \sum_{i=1}^{N} L(q_i) \tag{9}$$

By incorporating soft labels in this way, the model learns to approximate the probability distribution of neighbors across partitions, achieving consistency with the true distribution. This approach enables the model to assign higher probabilities to partitions with a denser neighbor distribution, improving recall performance as the model is more likely to retrieve partitions containing a larger proportion of relevant neighbors.

3.4 Repartitioning of boundary nodes

At the end of each training iteration, we perform a repartitioning step specifically for boundary nodes to address any imbalance issues. Using the trained MLP model, each boundary node n_j is reassigned to a partition based on the predicted probabilities $P(y|n_j)$, thus ensuring a more balanced distribution without affecting the stable partitions. For each boundary node n_j, we initially assign it to the partition with the highest predicted probability. If this partition has reached the limit $(1+\gamma)\frac{n}{m}$, the node is assigned to the next available partition with the highest probability. This can be expressed as:

$$p_{\text{new}} = \arg \max_{p \in P_{\text{available}}} P(p|n_j) \tag{10}$$

where $P_{\text{available}}$ denotes the set of partitions that have not reached their maximum capacity. This reassignment strategy for boundary nodes ensures that the overall partitioning structure remains stable, while achieving a more balanced distribution across partitions and enhancing the model's efficiency in handling nearest neighbor queries.

4 Experiments

Datasets. We used three real datasets of varying sizes and dimensions: SIFT (1M 128-dimensional image descriptors) [2], GIST (1M 960-dimensional global

image descriptors for scene recognition) [12], and GloVe (1.2M 100-dimensional word embeddings) [16].

Evaluation Metrics. We evaluate search efficiency in terms of distance computations, Query Per Second (QPS), and recall. For search effectiveness, we use Recall@M. Let $\mathcal{R}$ denote the retrieved neighbor set, and $\mathcal{T}$ the true neighbor set. Recall@M is defined as: Recall@M = $\frac{|\mathcal{R} \cap \mathcal{T}|}{M}$. To evaluate efficiency, we use unit query time and QPS metrics, allowing a comprehensive comparison of each method's effectiveness and speed. Additionally, we also compared edge cuts and balance, which reflect the quality of the partitions.

Baselines. To verify the effectiveness of our proposed method, we compared it with various graph partitioning algorithms, including METIS [10], KaHIP [17], Fennel [21], and LDG [19]. Additionally, we benchmarked our approach against other space partition-based methods, such as Neural LSH [6], PCA trees [18], and K-means [9]. Although Neural LSH also uses KaHIP for initial partitioning, it does not include our method's iterative boundary nodes repartitioning. To illustrate the impact of this refinement, we report its results separately.

4.1 Results

The experimental results, illustrated in the trade-off curve between Recall and Distance Computations in Fig. 3, demonstrate that our proposed method significantly outperforms existing methods in space partitioning tasks by achieving higher recall while maintaining efficiency. Compared to METIS, our method demonstrates a superior trade-off between Recall and Distance Computations. This improvement arises from our integration of a distance-aware multi-level graph partitioning strategy that explicitly considers space proximity, effectively reducing edge cuts and ensuring that closer neighbors are assigned to the same partition to enhance data locality. Unlike Neural LSH, which achieves reasonable recall but lacks dynamic repartitioning for partition boundaries, KaHIP combines the strengths of KaHIP for initial partitioning with a trained model to iteratively repartition boundary nodes. This dynamic boundary adjustment significantly improves partition quality by ensuring that boundary nodes are reassigned to optimal partitions, further boosting recall and demonstrating the effectiveness of our repartitioning strategy. In contrast, K-means and PCA trees, as traditional space partitioning methods, rely on centroid-based clustering or dimensionality reduction techniques to divide the space into regions. However, these methods often struggle to minimize edge cuts or dynamically optimize partitions based on space locality, resulting in relatively lower recall compared to approaches that leverage graph-based or distance-aware partitioning strategies. Additionally, LDG and Fennel, though lightweight and computationally efficient, fail to fully utilize the structural properties of the graph or incorporate distance information. This limitation leads to lower-quality partitions, making these methods less effective for high-dimensional datasets that require precise and locality-aware partitioning. The experimental results show that our method's distance-aware multi-level partitioning algorithm prioritizes closer neighbors during the partitioning process, ensuring that proximal nodes are assigned to the same partition,

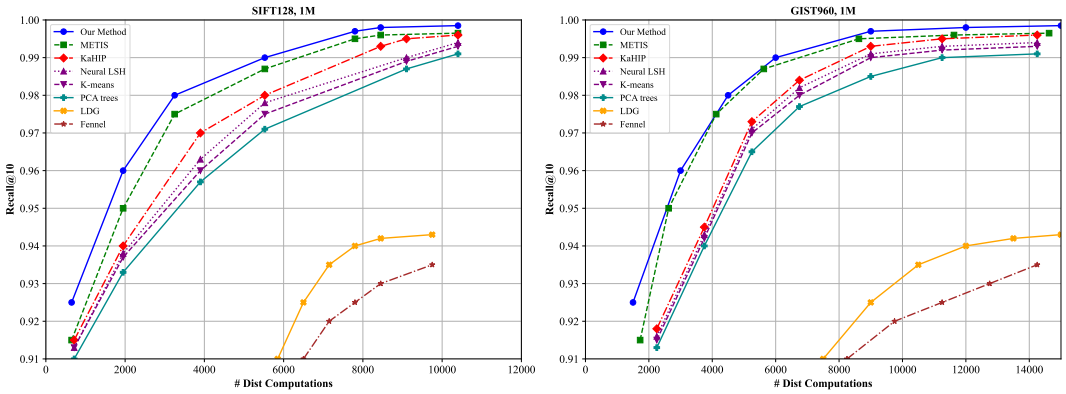

Fig. 3: Recall@10 V.S Distance computations.

while the trained model's refinement of boundary nodes further enhances partitioning quality and balance.

The Fig. 4 presents a detailed comparison of the experimental results of various methods across varying numbers of partitions and evaluation metrics, including distance computations, QPS, edge cuts, and partition balance. The Fig. 4(a) illustrates the number of actual distance computations required by each method, expressed as a percentage of K-means, to achieve a recall accuracy of 0.9. The figure clearly shows that our method consistently achieves the lowest distance computation ratio across various partition counts, outperforming all other methods. For example, Neural LSH and PCA trees require significantly more computations due to their less optimized partitioning strategies, resulting in higher computational overhead. While METIS and KaHIP perform better than PCA trees, they still fall short compared to our method, which leverages a distance-aware multi-level graph partitioning strategy. This strategy prioritizes closer neighbors while ensuring high recall, highlighting the efficiency of our approach in handling diverse partition sizes. The Fig. 4(b) displays the QPS performance of each method, again normalized to K-means. Our method, due to its superior partitioning quality, achieves significantly better QPS performance across all partition numbers, reflecting its ability to optimize data locality. In contrast, Neural LSH and LDG achieve relatively lower QPS, which highlights the limitations of their partitioning strategies in preserving locality. METIS performs reasonably well, but it falls short of our method due to its inability to explicitly consider spatial proximity during partitioning.

The Fig. 4(c) shows the number of edge cuts for different spatial partitioning methods. Edge cuts serve as a proxy for partitioning quality, as fewer edge cuts generally indicate better locality and grouping of similar data points. While METIS and KaHIP achieve low edge cuts, our method further reduces this metric by leveraging dynamic boundary node repartitioning in addition to distance-aware partitioning. Comparing Fig. 4(a) and Fig. 4(c), it is evident that edge

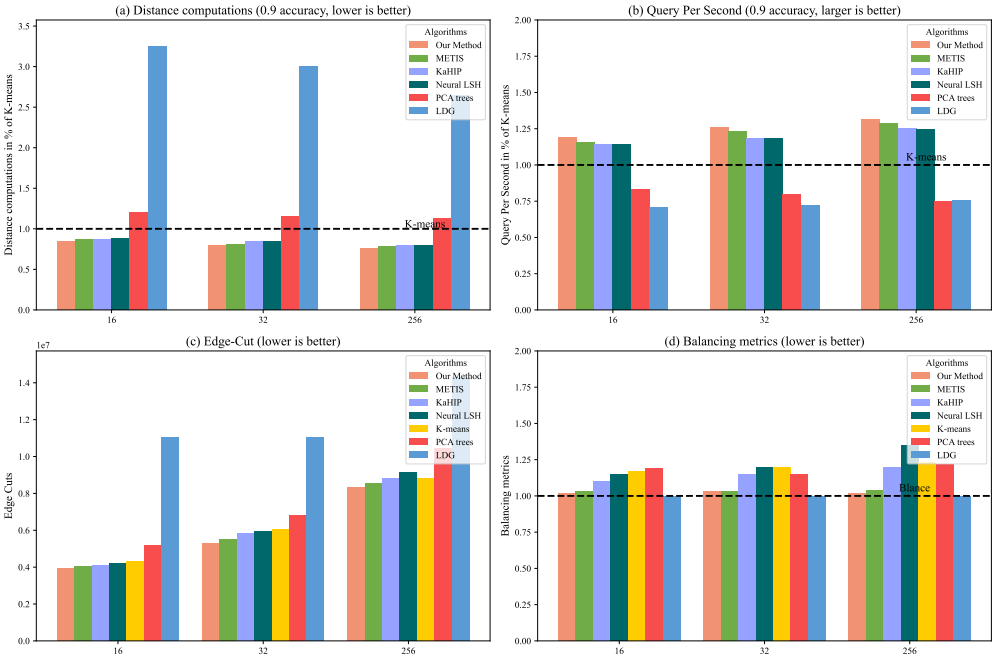

Fig. 4: Evaluation of Partitioning Methods Across Key Metrics: Distance computations, QPS, Edge-Cut, and Balance.

cuts partially reflect partitioning quality; however, at the same edge cut level, our method achieves better recall than METIS. This is due to our proposed distance-aware multi-level partitioning algorithm, which not only reduces edge cuts but also ensures that closer neighbors are more likely to reside in the same partition.

The Fig. 4(d) presents the balancing metrics for each method, where lower values indicate better balance across partitions. Our method achieves near-optimal balance, second only to LDG. However, LDG achieves this balance by assigning an equal number of data points to each partition, which results in poorer partitioning quality. To achieve the same level of recall accuracy as our method, LDG requires retrieving more data points across multiple partitions, undermining its efficiency. Our method outperforms traditional spatial partitioning methods such as K-means and PCA trees, which inherently struggle with balance and locality due to their reliance on centroid-based clustering or dimensionality reduction techniques. Furthermore, KaHIP performs better than Neural LSH, demonstrating that the trained model performs repartitioning of boundary nodes in our method effectively improves both partition quality and balance, achieving superior performance overall.

The Fig. 5 illustrates the convergence behavior of different methods on the GloVe dataset, focusing on the Recall@10 within the Top-1 partition. It is evi-

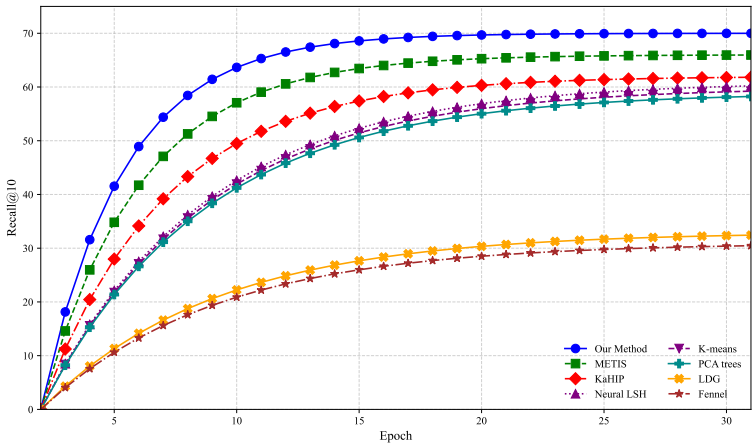

Fig. 5: Recall@10 V.S Epoch.

Table 1: Ablation Study: The Impact of Boundary Nodes Repartitioning

Partition Number	**Method**	**Recall@10 (Top-1)**	**Edge Cuts**	**Balance**
16	Full (Our Method)	76.18%	3923669	1.02
	w/o Repartitioning	75.35%	4028162	1.1
	Full (KaHIP)	73%	4200702	1.1
	Neural LSH	71.6%	4509272	1.15
32	Full (Our Method)	67.08%	5304871	1.03
	w/o Repartitioning	66.26%	5507374	1.1
	Full (KaHIP)	64.70%	5843302	1.15
	Neural LSH	63.86%	5944070	1.2
256	Full (Our Method)	51.96%	8329702	1.02
	w/o Repartitioning	50.82%	8523184	1.1
	Full (KaHIP)	48.76%	8829724	1.2
	Neural LSH	46.98%	9013258	1.22

dent from the Fig. 5 that our method demonstrates the fastest convergence and achieves the highest recall among all baselines. This indicates that a good initial partition can significantly enhance both the efficiency and quality of the training process. Additionally, our method repartitions boundary nodes after each training iteration, further improving partition quality and balance. This repartitioning process reduces the dispersion of neighbors across partitions, leading to higher recall and better overall performance. The Fig. 5 clearly demonstrates that methods like METIS and KaHIP, while effective, converge more slowly and achieve lower recall compared to our approach, primarily because they focus solely on minimizing edge cuts without considering the distances between nodes. Traditional methods like K-means and PCA trees also lag behind, as they fail to optimize locality and balance effectively during training.

The results in Table 1 demonstrate the effectiveness of our proposed method, particularly the dynamic boundary node repartitioning mechanism, in improving partitioning quality and performance across different partition numbers (16, 32, 256). Our method consistently achieves the highest Recall@10 within the Top-1 partition, with values of 76.18%, 67.08%, and 51.96% for 16, 32, and 256 partitions, respectively, outperforming the configurations without repartitioning, Full KaHIP (Neural LSH w/Repartitioning), and Neural LSH. For instance, at 256 partitions, our method achieves a Recall@10 improvement of 1.14% over the w/o Repartitioning configuration and 4.98% over Neural LSH. This improvement is coupled with a significant reduction in edge cuts, with our method achieving 8329702 edge cuts at 256 partitions, compared to 8829724 for KaHIP and 9013258 for Neural LSH, highlighting its ability to enhance data locality. Furthermore, our method maintains near-optimal balance metrics close to 1.02 across all partition numbers, outperforming Neural LSH (1.15–1.22) and KaHIP (1.1–1.2). The comparison between Full (Our Method) and w/o Repartitioning underscores the critical role of dynamic boundary node repartitioning, which clusters neighbors into the same partitions, thereby boosting recall and ensuring balanced partitioning.

5 Conclusions

In this paper, we propose a novel approach that integrates a distance-aware multilevel partitioning algorithm with a lightweight model to address challenges in nearest neighbor search. Initially, our multilevel partitioning algorithm incorporates node distance information to create balanced and meaningful partitions that preserve the local structure of the data. Following this, we deploy a lightweight classifier to predict the partition where the neighbors of a query point are most likely to be located. To address the issue of query neighbors being dispersed across partitions, we use soft labels during training, assigning probabilities to partitions based on neighbor density and optimizing the KL divergence between the predicted and ground truth distributions to align the model's output with the actual neighbor distribution and maximize recall. Additionally, recognizing potential imbalances in the initial partitioning, we introduce an iterative repartitioning step where boundary nodes are dynamically repartitioned based on the classifier's predictions. A series of results verify the effectiveness of the proposed method, which outperforms the baselines in both search efficiency (distance computations, Query Per Second (QPS), and recall) and partitioning quality (edge cuts and balance).

Acknowledgements This work was supported by NSFC grant (No. 62136002 and 62477014), Ministry of Education Research Joint Fund Project (8091B042239), and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

References

1. Andoni, A., Naor, A., Nikolov, A., Razenshteyn, I.P., Waingarten, E.: Data-dependent hashing via nonlinear spectral gaps. Proc. ACM SIGACT Symp. Theory Comput. (2018)
2. Aumüller, M., Bernhardsson, E., Faithfull, A.J.: Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. Inf. Syst. **87** (2018)
3. Bentley, J.L.: Multidimensional binary search trees in database applications. IEEE Trans. Softw. Eng. **SE-5**, 333–340 (1979)
4. Chung, F.R.K.: Laplacians of graphs and cheeger inequalities. Combinatorica (1993)
5. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. Proc. ACM Symp. Comput. Geom. (2004)
6. Dong, Y., Indyk, P., Razenshteyn, I.P., Wagner, T.: Learning space partitions for nearest neighbor search. IEEE Data Eng. Bull. **46**, 55–68 (2019)
7. Gottesbüren, L., Dhulipala, L., Jayaram, R., Lacki, J.: Unleashing graph partitioning for large-scale nearest neighbor search. ArXiv **abs/2403.01797** (2024)
8. Gupta, G., Medini, T., Shrivastava, A., Smola, A.J.: Bliss: A billion scale index using iterative re-partitioning. Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (2022)
9. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33**, 117–128 (2011)
10. Karypis, G., Kumar, V.: Parallel multilevel k-way partitioning scheme for irregular graphs. Proc. ACM/IEEE Conf. Supercomput. pp. 35–35 (1996)
11. Li, W., Feng, C., Lian, D., et al.: Learning balanced tree indexes for large-scale vector retrieval. Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (2023)
12. Manohar, M.D., Shen, Z., et al.: Parlayann: Scalable and deterministic parallel graph-based approximate nearest neighbor search algorithms. SIGPLAN (2023)
13. Omohundro, S.M.: Five balltree construction algorithms (2009)
14. Pan, J.J., Wang, J., Li, G.: Survey of vector database management systems. VLDB J. **33**, 1591–1615 (2023)
15. Pan, J.J., Wang, J., Li, G.: Vector database management techniques and systems. Proc. Companion Int. Conf. Manage. Data (2024)
16. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. Proc. Conf. Empir. Methods Nat. Lang. Process. (2014)
17. Sanders, P., Schulz, C.: Think locally, act globally: Highly balanced graph partitioning. The Sea (2013)
18. Sproull, R.F.: Refinements to nearest-neighbor searching ink-dimensional trees. Algorithmica **6**, 579–589 (1991)
19. Stanton, I., Kliot, G.: Streaming graph partitioning for large distributed graphs. Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (2012)
20. Tian, Y., Yan, T.K., Zhao, X., et al.: A learned index for exact similarity search in metric spaces. IEEE Trans. Knowl. Data Eng. **35**, 7624–7638 (2022)
21. Tsourakakis, C.E., Gkantsidis, C., Radunovic, B., Vojnović, M.: Fennel: streaming graph partitioning for massive scale graphs. Proc. ACM Int. Conf. Web Search Data Min. (2014)
22. Wang, J., Zhang, T., Song, J., et al.: A survey on learning to hash. IEEE Trans. Pattern Anal. Mach. Intell. **40**, 769–790 (2016)
23. Zeng, Y., Tong, Y., Chen, L.: Litehst: A tree embedding based method for similarity search. Proc. ACM Manage. Data **1**, 1 – 26 (2023)
24. Zhang, H., Wang, Y., Chen, Q., et al.: Model-enhanced vector index. Advances in Neural Information Processing Systems **36** (2024)