Jointly Learning Graph Partitions and Classifiers for Fast Nearest Neighbor Search

Soham Samal, Thea Wenyi Zhu, Roberto Brera
Columbia University

Abstract

Graph-based Approximate Nearest Neighbor (ANN) methods such as Neural LSH[1] proceed in two distinct phases: (1) construct a k-NN graph over the dataset and compute a balanced graph partition[2], and (2) train a classifier that extends this partition to the ambient space $\mathbb{R}^d$. This two-stage pipeline is suboptimal: the partition is optimized in graph space without considering the classifier's ability to represent it, while the classifier is constrained to imitate a partition it did not help shape. A natural question arises (explicitly posed in prior work): *can we jointly optimize both the graph partition and the classifier so that the two components adapt to one another?*[1] If successful, such a joint formulation could produce partitions that are both graph-friendly (low cut and smoothness) and classifier-friendly (easy to separate in $\mathbb{R}^d$), potentially improving end-to-end ANN performance.

In this project, we explore two attempts at designing such a joint method:

1. a relaxed, fully differentiable partitioner using soft assignments, and
2. a structured, alternating optimization scheme using hard labels with classifier updates.

We report both negative and positive findings.

1 Introduction

The problem of Approximate Nearest Neighbor Search (ANNS) [3] in high-dimensional spaces is a fundamental challenge in algorithms, machine learning, and data retrieval. Given a dataset $P \subset \mathbb{R}^d$, the goal is to preprocess P so that, for a query point $q \in \mathbb{R}^d$, one can efficiently retrieve a point whose distance to q is within a small factor of the true nearest neighbor distance. Exact nearest neighbor search suffers from the curse of dimensionality, motivating the development of approximate methods with provable efficiency guarantees.

1.1 Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH)[4] provides one of the most influential algorithmic frameworks for ANNS. An LSH family consists of randomized hash functions with the property that nearby points collide with higher probability than distant points. By combining multiple such hash functions, LSH constructs a randomized partition of the space into buckets, enabling sublinear-time query algorithms under appropriate distance measures.

The strength of classical LSH lies in its rigorous theoretical guarantees: under suitable assumptions on the metric space, LSH-based data structures achieve provable tradeoffs between query time, space , and approximation. However, the hash functions are drawn from fixed, distribution-dependent families and are largely agnostic to the specific structure of the dataset. As a result, classical LSH often exhibits suboptimal empirical performance on real-world, highly structured data.

1.2 Data-Dependent and Learned Hashing

To address the limitations of data-independent hashing, a broad line of work has explored data-dependent and learned hashing schemes[5]. These methods adapt the partitioning of the space to the distribution of the data, often yielding substantial empirical improvements.

Neural Locality-Sensitive Hashing (Neural LSH)[1] Rather than learning hash functions directly, Neural LSH first constructs a k-nearest neighbor graph over the dataset and computes a balanced partition of this graph using a high-quality graph partitioning algorithm known as KaHIP[2]. A classifier is then trained to predict the resulting partition labels, thereby learning a routing function from $\mathbb{R}^d$ to partition blocks.

In this approach, graph locality is enforced through explicit partitioning of the k-NN graph, while efficient query routing is achieved through supervised learning. Neural LSH has been shown to achieve strong empirical performance on standard nearest neighbor benchmarks while retaining predictable candidate set sizes through balanced partitions.

1.3 Limitations and Motivation

Despite its effectiveness, Neural LSH adopts a strictly two-stage pipeline in which the partition is fixed before classifier training. This separation can lead to misalignment between the graph-induced partition and the geometric decision boundaries realizable by the classifier. In particular, a partition that is optimal with respect to graph cut objectives may be unnecessarily complex or fragmented when viewed in the original feature space, complicating the routing task.

This observation motivates the central question of this work: *can the space partition and routing classifier be jointly optimized so that each adapts to the other?*[1] By allowing the partition to evolve in response to classifier predictions, one may obtain partitions that better respect both graph locality and classifier learnability, leading to improved nearest neighbor retrieval performance.

In this work, we explore this question through a fully differentiable soft assignment and alternating refinement framework that iteratively updates the partition and classifier.

2 Intuition

Let $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ be a dataset and let G be its k-NN graph. Instead of computing a discrete partition $\pi : P \to [m]$ and then training a classifier to mimic it, we propose to learn a single function

$$g_\theta : \mathbb{R}^d \to \Delta^{m-1},$$

where $g_\theta(x)$ is a probability distribution over m bins. For each data point p_i, the vector $a_i = g_\theta(p_i)$ defines a *soft* assignment to bins.

The goal is to learn θ such that:

1. Nearby points in the k-NN graph have similar assignments.
2. The bins are roughly balanced: $\sum_i a_{i,c} \approx n/m$.
3. The induced hard partition (via $\arg\max_c a_{i,c}$) yields efficient multi-probe NNS for unseen queries.

In this approach, the graph partition *is* the classifier. No separate preprocessing partitioning step is required.

3 Continuous Relaxation via Soft Assignment

3.1 Model & Loss Functions

We first attempted to define a partition of the dataset through a parametric map $g_\theta : \mathbb{R}^d \to \Delta^{m-1}$, implemented as an MLP with softmax outputs. For each point x_i, the model produces a probability vector $a_i = g_\theta(x_i)$ interpreted as a soft bin assignment.

Given a k-NN graph $G = (V, E)$ over the dataset, we defined the following losses:

Soft Graph Cut The expected probability that edge (i, j) is cut is

$$\ell_{\text{cut}}(i, j) = 1 - \sum_{c=1}^{m} a_{i,c} a_{j,c}.$$

[6, 7]The full cut loss is the average over edges.

Balance Let $s_c = \frac{1}{n} \sum_i a_{i,c}$ denote the mass of bin c. The balance penalty is

$$\ell_{\text{bal}} = \frac{1}{m} \sum_{c=1}^{m} \left(s_c - \frac{1}{m} \right)^2.$$

Smoothness A Laplacian-style regularizer encourages neighboring nodes to have similar assignments:

$$\ell_{\text{smooth}} = \frac{1}{|E|} \sum_{(i,j) \in E} \|a_i - a_j\|_2^2.$$

[7]

Entropy Regularization To prevent one-hot assignments, we added this as a fine-tuning adjustment

$$\ell_{\text{ent}} = 1 - \frac{1}{\log m} H(a_i), \qquad H(a_i) = -\sum_{c=1}^{m} a_{i,c} \log a_{i,c}.$$

The final objective was

$$\mathcal{L} = \lambda_{\text{cut}} \ell_{\text{cut}} + \lambda_{\text{bal}} \ell_{\text{bal}} + \lambda_{\text{smooth}} \ell_{\text{smooth}} + \lambda_{\text{ent}} \ell_{\text{ent}}.$$

3.2 Experimental Setup

We worked with the `SIFT1M` dataset (1M points, 128 dimensions) and constructed a $k = 10$ nearest-neighbor graph over a set of $1,000,000$ points. We used $m = 64$ bins and trained the MLP with Adam. We monitored bin masses, cut/smooth losses, entropy, and total loss.

3.3 Observed Failure Mode: Total Collapse into a Single Bin

Despite extensive tuning (increasing balance weight, adding entropy, temperature scaling), the continuous relaxation consistently collapsed to the degenerate solution:

$$a_i \approx e_{c^*} \quad \text{for all } i,$$

which means all points were assigned to the same bin.

Representative training logs:

```
Epoch 001: max bin mass = 0.689
Epoch 010: max bin mass = 0.690
Epoch 025: max bin mass = 0.959
Epoch 030: max bin mass = 0.9997
Epoch 050: max bin mass = 1.0000
```

As the model collapsed:

$$\ell_{\text{cut}} \to 0, \quad \ell_{\text{smooth}} \to 0, \quad \ell_{\text{ent}} \to 1, \quad \ell_{\text{bal}} \to \tfrac{1}{m}.$$

The dominant effect is that placing all points in one bin minimizes both cut and smoothness perfectly, while the balance and entropy terms provide insufficient opposing gradient once the softmax saturates (vanishing gradients). In practice, the model enters a basin where gradients from all relevant terms vanish, making recovery impossible.

4 Alternating Refinement

4.1 Problem Setup

Let $P = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ be a dataset of points. We construct an approximate k-nearest neighbor graph $G = (V, E), V = \{1, \ldots, n\}$ where an undirected edge $(i, j) \in E$ indicates that x_j is among the k nearest neighbors of x_i under the Euclidean metric.

We seek a partition of V into m blocks, represented by a labeling function $b : V \to \{1, \ldots, m\}$. Let $P_c = \{i \in V : b(i) = c\}$ denote the set of vertices assigned to block c.

The partition is required to satisfy a balance constraint: for fixed parameters $0 < \varepsilon < 1$,

$$(1 - \varepsilon)\frac{n}{m} \;\le\; |P_c| \;\le\; (1 + \varepsilon)\frac{n}{m} \quad \text{for all } c \in \{1, \ldots, m\}.$$

In addition to the partition, we consider a classifier

$$f_\theta : \mathbb{R}^d \to \Delta^{m-1},$$

parameterized by θ, where Δ^{m-1} denotes the probability over m labels. For a point x_i, the classifier outputs a distribution $p_\theta(\cdot \mid x_i)$ over blocks, which is used to route both data points during training and queries at search time.

4.2 Optimization Perspective

The partition b and classifier parameters θ are chosen to satisfy three competing objectives:

1. **Graph locality:** vertices connected by edges in G should be assigned to the same block;
2. **Balance:** all blocks should have comparable cardinality;
3. **Learnability:** the labeling b should be predictable by the classifier f_θ.

These requirements can be viewed as inducing a joint optimization problem over a discrete variable b and continuous parameters θ. Direct optimization is intractable due to the combinatorial nature of balanced graph partitioning, as seen from the first experiment. Instead, we adopt an alternating optimization strategy in which b and θ are updated in turn while holding the other fixed.

This approach can be interpreted as block-coordinate ascent on a proxy objective, analogous to EM-style procedures involving discrete latent variables and continuous model parameters[8, 9].

4.3 Initialization

We initialize the labeling $b^{(0)}$ by applying a balanced graph partitioning algorithm[2] to the k-nearest neighbor graph G. This produces an initial partition that approximately maximizes graph locality subject to balance constraints. Given this initial labeling, we train a classifier $f_{\theta^{(0)}}$ to predict the block assignments.

This initialization provides a strong locality-preserving starting point and avoids the instability associated with random partitions.

4.4 Alternating Refinement Procedure

Starting from $(b^{(0)}, \theta^{(0)})$, we perform a sequence of refinement iterations indexed by $t = 0, 1, 2, \ldots$. Each iteration consists of the following two steps.

Classifier Update. With the partition $b^{(t)}$ fixed, we update the classifier parameters by solving the supervised learning problem

$$\theta^{(t+1)} \in \arg\max_{\theta} \sum_{i=1}^{n} \log p_\theta\big(b^{(t)}(i) \mid x_i\big).$$

This step optimizes the likelihood of the current labeling under the classifier and encourages f_θ to approximate the partition boundaries.

Partition Update. With the classifier parameters $\theta^{(t+1)}$ fixed, we update the partition through a sequence of local reassignment moves. For a vertex $i \in V$, let

$$\hat{b}_i = \arg\max_{c \in \{1,\ldots,m\}} p_{\theta^{(t+1)}}(c \mid x_i)$$

denote the classifier's most confident prediction. We consider reassigning vertex i from its current block $b^{(t)}(i)$ to $\hat{b}_i$.

A reassignment is accepted only if it satisfies the balance constraints and yields a strict improvement in graph locality, measured by the number (or total weight) of neighbors in G that share the same block. This update rule constitutes a local search step on the space of feasible balanced partitions, biased toward classifier-consistent assignments.

4.5 Termination and Output

The refinement procedure is repeated until no admissible local reassignment yields further improvement. Since each accepted move strictly improves a local objective and the set of feasible balanced partitions is finite, the procedure terminates after finitely many steps.

The final output consists of a refined partition $b^\star$ and classifier parameters $\theta^\star$. At query time, a query point $q \in \mathbb{R}^d$ is routed by evaluating $p_{\theta^\star}(\cdot \mid q)$, probing a small number of highest-ranked blocks, and performing exact distance computations within the resulting candidate set. The query algorithm itself is unchanged from the Neural LSH framework; only the learned partition and routing function differ.

5 Theoretical Guarantees

We formalize the alternating refinement procedure as a local-search on a discrete feasible set and prove (i) finite termination, (ii) local optimality with respect to an admissible move set, and (iii) monotonicity of an explicit potential under idealized alternating updates. Balanced graph partitioning is NP-hard, so we do not claim global optimality.

Let $G = (V, E)$ be a (weighted) graph on $V = \{1, \dots, n\}$ with edge weights $w_{ij} \geq 0$ for $(i, j) \in E$. A partition into m blocks is represented by a labeling $b \in [m]^V, b_i \in \{1, \dots, m\}$. Let $P_c(b) = \{i \in V : b_i = c\}$ be the c-th block and $s_c(b) = |P_c(b)|$ its size.

Definition 1. *Given $L \leq U (L = (1-\varepsilon)n/m, U = (1+\varepsilon)n/m)$, Define the set of all feasible balanced partitions*

$$\mathcal{B} \;=\; \{b \in [m]^V : \forall c \in [m],\; L \leq s_c(b) \leq U\},.$$

Definition 2. *Define the within-block cohesion (negative cut) objective*

$$\Phi(b) \;=\; \sum_{(i,j)\in E} w_{ij}\, \mathbf{1}[b_i = b_j].$$

Equivalently, $\Phi(b)$ is the total weight of edges whose endpoints lie in the same block.

Definition 3 (Admissible single-vertex move)**.** *For $b \in \mathcal{B}$, vertex $i \in V$, and label $c \in [m]$, define $b^{(i\to c)}$ to be the labeling obtained from b by setting $b_i \leftarrow c$ and leaving all other labels unchanged. We say the move $(i \to c)$ is feasible if $b^{(i\to c)} \in \mathcal{B}$.*

The refinement step considers a restricted set of feasible moves (only to a classifier-suggested label and only above some margin threshold), but the results below hold for any rule that only accepts moves which strictly increase the chosen potential.[10]

Definition 4. *For a fixed labeling b, define the local within-block weight of vertex i to block c by*

$$W_i(c; b) \;=\; \sum_{j:(i,j)\in E} w_{ij}\, \mathbf{1}[b_j = c].$$

This is the total weight from i to neighbors currently labeled c.

Lemma 1 (Change in Cohesion Under a Single-Vertex Move)**.** *Let $b \in [m]^V$, $i \in V$, and $c \in [m]$. Then*

$$\Phi\big(b^{(i\to c)}\big) - \Phi(b) \;=\; W_i(c; b) - W_i(b_i; b).$$

Proof. Only edges incident to i can change their contribution to Φ when we relabel i. For any neighbor j of i, the edge (i, j) contributes w_{ij} to Φ if and only if the labels match. Before the move, (i, j) contributes $w_{ij}\mathbf{1}[b_j = b_i]$. After the move, it contributes $w_{ij}\mathbf{1}[b_j = c]$. Summing the difference over all neighbors j yields

$$\Phi\big(b^{(i\to c)}\big) - \Phi(b) = \sum_{j:(i,j)\in E} w_{ij}\,(\mathbf{1}[b_j = c] - \mathbf{1}[b_j = b_i]) = W_i(c; b) - W_i(b_i; b).$$

□

Lemma 2 (Boundedness of Cohesion)**.** *For all $b \in [m]^V$,*

$$0 \;\le\; \Phi(b) \;\le\; \sum_{(i,j)\in E} w_{ij}.$$

Proof. Each term $w_{ij}\mathbf{1}[b_i = b_j]$ is nonnegative and at most w_{ij}. Summing over edges gives the statement above. □

We now prove that any refinement procedure that repeatedly applies strictly improving feasible moves must terminate in finitely many steps at a 1-move local optimum.

Theorem 1 (Finite Termination of Strictly Improving Refinement)**.** *Consider any iterative refinement procedure that maintains $b^{(t)} \in \mathcal{B}$ and, at each step, either stops or selects a feasible move $(i \to c)$ such that*

$$\Phi\big(b^{(t+1)}\big) \;>\; \Phi\big(b^{(t)}\big), \quad \text{where } b^{(t+1)} := (b^{(t)})^{(i\to c)}.$$

Then the procedure terminates after finitely many steps.

Proof. The feasible set $\mathcal{B}$ is finite because it is a subset of $[m]^V$, which has size m^n. Since Φ strictly increases at every accepted move, the sequence $\{\Phi(b^{(t)})\}$ is strictly increasing. Hence the procedure cannot revisit any labeling (doing so would repeat a previous Φ value). Because there are only finitely many feasible labelings, only finitely many strict increases are possible. Thus, the procedure must terminate. □

Definition 5 (1-Move Local Optimality)**.** *A labeling $b^\star \in \mathcal{B}$ is a* (feasible) 1-move local optimum *for Φ if for every vertex $i \in V$ and every label $c \in [m]$ such that $b^{\star(i\to c)} \in \mathcal{B}$,*

$$\Phi\big(b^{\star(i\to c)}\big) \;\le\; \Phi(b^\star).$$

Theorem 2 (Local Optimality of Terminal Partition)**.** *Assume the refinement procedure of Theorem 1 terminates at $b^\star \in \mathcal{B}$ because no admissible move yields a strict increase in Φ. Then $b^\star$ is a feasible 1-move local optimum for Φ with respect to the move set considered by the procedure. If the procedure considers* all *feasible single-vertex moves, then $b^\star$ is a feasible 1-move local optimum in the sense of the above definition.*

Proof. If the algorithm terminates, by definition there is no admissible feasible move $(i \to c)$ from $b^\star$ with $\Phi(b^{\star(i\to c)}) > \Phi(b^\star)$. Thus $\Phi(b^{\star(i\to c)}) \le \Phi(b^\star)$ for every admissible feasible move, which is exactly 1-move local optimality restricted to that move set. If all feasible moves are admissible, the statement holds for every feasible single-vertex move. □

Lemma 1 shows that checking whether a move improves Φ depends only on the neighborhood of i in G. To connect refinement to joint optimization with a classifier, we introduce a joint function.

Definition 6. *Let f_θ be a probabilistic classifier producing probabilities $p_\theta(c \mid x_i)$ over labels $c \in [m]$. Define the* joint potential

$$\Psi(b, \theta) \;=\; \Phi(b) \;+\; \lambda \sum_{i=1}^{n} \log p_\theta(b_i \mid x_i), \qquad \lambda \ge 0,$$

with the balance constraint enforced by restricting $b \in \mathcal{B}$.

Lemma 3 (Classifier Update is Non-Decreasing in Ψ)**.** *Fix $b \in \mathcal{B}$. Let θ^+ satisfy*

$$\sum_{i=1}^{n} \log p_{\theta^+}(b_i \mid x_i) \;\geq\; \sum_{i=1}^{n} \log p_{\theta}(b_i \mid x_i).$$

Then $\Psi(b, \theta^+) \geq \Psi(b, \theta)$.

Proof. When b is fixed, $\Phi(b)$ is constant with respect to θ. Thus

$$\Psi(b, \theta^+) - \Psi(b, \theta) = \lambda \left(\sum_{i=1}^{n} \log p_{\theta^+}(b_i \mid x_i) - \sum_{i=1}^{n} \log p_{\theta}(b_i \mid x_i) \right) \geq 0.$$

□

Lemma 4 (Partition Refinement is Non-Decreasing in Ψ)**.** *Fix θ. Suppose a refinement step changes b to b^+ (with $b, b^+ \in \mathcal{B}$) such that*

$$\Psi(b^+, \theta) \;>\; \Psi(b, \theta).$$

Then the joint potential strictly increases under this step.

Proof. This is immediate. □

Theorem 3 (Monotonicity and Finite Termination of Ideal Alternating Ascent)**.** *Consider an alternating procedure producing $(b^{(t)}, \theta^{(t)})$ such that:*

1. *(*Classifier step*) $b^{(t)}$ is fixed and $\theta^{(t+1)}$ satisfies the condition of Lemma 3.*
2. *(*Partition step*) $\theta^{(t+1)}$ is fixed and $b^{(t+1)} \in \mathcal{B}$ is chosen so that $\Psi(b^{(t+1)}, \theta^{(t+1)}) > \Psi(b^{(t)}, \theta^{(t+1)})$, unless no such update is possible (in which case we stop).*

Then $\Psi(b^{(t)}, \theta^{(t)})$ is monotonically non-decreasing over iterations, and the procedure terminates in finitely many partition updates.

Proof. By Lemma 3, the classifier step does not decrease Ψ. By construction, the partition step strictly increases Ψ unless the algorithm stops. Since b ranges over the finite set $\mathcal{B}$ and the partition step strictly increases Ψ while keeping θ fixed, the algorithm cannot perform infinitely many strict partition improvements without revisiting a labeling. This means only finitely many strict partition updates are possible and the procedure terminates. Monotonicity comes directly by combining the two non-decreasing updates. □

Note Theorem 3 formalizes the alternating refinement as a coordinate-ascent method on a clear potential, under the idealization that each step improves the relevant sub-objective. In practice, classifier training and partition updates are approximate and restricted (e.g. thresholded moves), so the theorem should be interpreted as an explanatory guarantee rather than a global optimality result.

We emphasize that these results do not imply:

- approximation guarantees for the globally optimal balanced partition,
- worst-case ANN query-time/recall guarantees of LSH type,
- or convergence to a global maximizer of Ψ.

They do, however, provide formal justification that the refinement component is a well-defined local-search procedure with termination and local-optimality properties, and that the overall method can be viewed as monotone improvement of a joint potential under idealized alternating ascent.

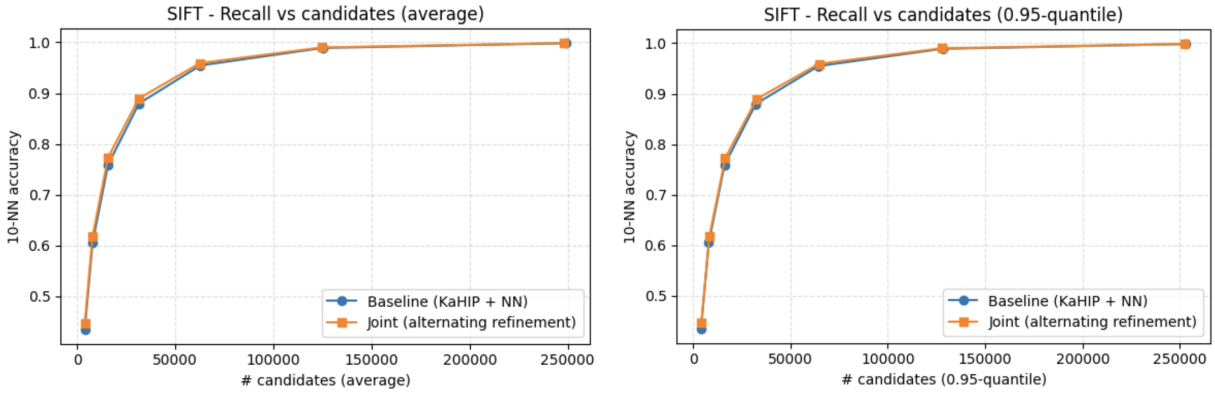

Figure 1: Training results of baseline (Neural LSH) and our proposed alternating refinement variant.

6 Results

We evaluate the proposed joint optimization framework on the SIFT1M benchmark and compare it against the baseline Neural LSH pipeline. Performance is measured using the standard recall-candidate tradeoff adopted in prior work, which captures both average query efficiency and worst-case behavior.

6.1 Evaluation Metrics

Following Neural LSH, we report k-nearest neighbor accuracy with $k = 10$ as a function of the number of distance computations (candidates) performed at query time. We measure the average number of candidates per query, and the 0.95-quantile number of candidates, which reflects latency.

6.2 Joint Optimization via Alternating Refinement

Figure 1 presents our results on SIFT1M comparing the baseline Neural LSH pipeline (KaHIP partition followed by classifier training) with the proposed joint optimization scheme based on alternating refinement.

Across both evaluation metrics, the joint method consistently matches or improves upon the baseline. In particular:

- For a fixed number of candidates, the joint method achieves higher 10-NN accuracy.
- For a fixed target recall level, the joint method requires fewer candidates on average.
- Improvements are also observed at the 0.95-quantile, indicating better worst-case query behavior.

These gains are most pronounced in the intermediate regime, where routing errors dominate performance. This suggests that the refinement procedure effectively reduces misrouted queries by improving alignment between the partition structure and the classifier's geometric decision boundaries.

7 Conclusion

The observed improvements support the central hypothesis of this work: allowing the partition and classifier to co-adapt leads to partitions that better respect both graph locality and classifier learnability. While the initial KaHIP partition already preserves nearest-neighbor structure, alternating refinement selectively adjusts boundary points whose reassignment improves local graph coherence and classifier confidence.

Importantly, the query procedure itself is unchanged from Neural LSH. All gains arise solely from improved partition quality and routing accuracy, highlighting the benefit of joint optimization even under identical query-time constraints.

Overall, the results demonstrate that the proposed joint optimization framework yields a strictly better recall-candidate tradeoff on SIFT1M than the original two-stage Neural LSH pipeline. These findings suggest that classifier-aware partition refinement is a promising direction for learned nearest neighbor indexing, particularly in high-dimensional regimes where routing accuracy is critical.

7.1 Note

GPT-5 assisted in summarization and polishing of the report, as well as proof thinking.

References

[1] Yihe Dong et al. "Neural Locality-Sensitive Hashing". In: *Advances in Neural Information Processing Systems* 33 (2020).

[2] Peter Sanders and Christian Schulz. "Think locally, act globally: Highly balanced graph partitioning". In: *Proceedings of the 12th International Symposium on Experimental Algorithms* (2013), pp. 164–175.

[3] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: Towards removing the curse of dimensionality". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (1998), pp. 604–613.

[4] Alexandr Andoni and Piotr Indyk. "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions". In: *Communications of the ACM* 51.1 (2008), pp. 117–122.

[5] Alexandr Andoni et al. "Optimal data-dependent hashing for approximate nearest neighbor search". In: *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing* (2015), pp. 793–801.

[6] Jianbo Shi and Jitendra Malik. "Normalized Cuts and Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8 (2000), pp. 888–905.

[7] Ulrike von Luxburg. "A Tutorial on Spectral Clustering". In: *Statistics and Computing* 17.4 (2007), pp. 395–416.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[9] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B* 39.1 (1977), pp. 1–22.

[10] Brian W. Kernighan and Shen Lin. "An efficient heuristic procedure for partitioning graphs". In: *Bell System Technical Journal* 49.2 (1970), pp. 291–307.