FARGO: Fast Maximum Inner Product Search via Global Multi-Probing

Xi Zhao[1], Bolong Zheng[1], Xiaomeng Yi[2], Xiaofan Luan[2], Charles Xie[2], Xiaofang Zhou[3], Christian S. Jensen[4]
[1]Huazhong University of Science and Technology, Wuhan, China [2]Zilliz, Shanghai, China,
[3]Hong Kong University of Science and Technology, Hong Kong, China [4]Aalborg University, Aalborg, Denmark
{zhaoxi,bolongzheng}@hust.edu.cn
{xiaomeng.yi,xiaofan.luan,charles.xie}@zilliz.com
zxf@cse.ust.hk,csj@cs.aau.dk

ABSTRACT

Maximum inner product search (MIPS) in high-dimensional spaces has wide applications but is computationally expensive due to the curse of dimensionality. Existing studies employ asymmetric transformations that reduce the MIPS problem to a nearest neighbor search (NNS) problem, which can be solved using locality-sensitive hashing (LSH). However, these studies usually maintain multiple hash tables and locally examine them one by one, which may cause additional costs on probing unnecessary points. In addition, LSH is applied without taking into account the properties of the inner product. In this paper, we develop a fast search framework FARGO for MIPS on large-scale, high-dimensional data. We propose a global multi-probing (GMP) strategy that exploits the properties of the inner product to globally examine high quality candidates. In addition, we develop two optimization techniques. First, different with existing transformations that introduce either distortion errors or data distribution imbalances, we design a novel transformation, called random XBOX transformation, that avoids the negative effects of data distribution imbalances. Second, we propose a global adaptive early termination condition that finds results quickly and offers theoretical guarantees. Extensive experiments with real-world data offer evidence that FARGO is capable of outperforming existing proposals in terms of both accuracy and efficiency.

PVLDB Reference Format:
Xi Zhao, Bolong Zheng, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, Christian S. Jensen. FARGO: Fast Maximum Inner Product Search via Global Multi-Probing. 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 https://github.com/Jacyhust/FARGO_VLDB23.

1 INTRODUCTION

Given a query point $q \in \mathbb{R}^d$ and a dataset $\mathcal{D} \subseteq \mathbb{R}^d$, the maximum inner product search (MIPS) problem aims to find a point x in $\mathcal{D}$ that has the maximum inner product $q^\top x$. The MIPS problem in high-dimensional spaces is core functionality in a wide variety of applications, such as matrix factorization based recommendation [1, 34], multi-class label prediction [11, 21], similar-item retrieval [44], structural SVM [22], and deep learning [17], to name but a few.

A basic solution to the MIPS problem is the space partitioning tree-based method [23, 24, 36], which works well in low dimensional spaces. However, when the dimensionality increases to several hundred or higher, the search time increases exponentially [2, 6, 8]. A more efficient alternative is to find the approximate MIPS result. If we consider the inner product as a similarity metric, the approximate MIPS problem is similar to the approximate nearest neighbor search (NNS). As locality-sensitive hashing (LSH) has demonstrated its superiority on solving approximate NNS [4, 25, 27, 42, 46, 47], it is natural to apply LSH for approximate MIPS [4, 25, 27, 42, 46, 47]. However, to work correctly, LSH requires that the metric satisfies the condition that for any point q, the point that has the largest similarity to q is q itself. Unfortunately, the inner product does not satisfy this condition. That is, a point x always exists that is different from q and such that $q^\top x > q^\top q$.

Existing studies adopt asymmetric transformations that convert the MIPS problem into NNS, and then use LSH to solve the NNS problem. Specifically, they employ the basic (K, L)-bucketing [3] to construct hash tables, and they use Multi-Probe [31] that maintains a local probing sequence for each hash table. For each hash table, they examine the candidates from “more promising" hash buckets to “less promising" ones based on the local probing sequence. This process is repeated in all hash tables so that many “less promising" hash buckets in different hash tables are unnecessarily examined, which may cause additional costs and affect the query performance. We develop a fast search framework FARGO for MIPS on large-scale, high-dimensional data. Unlike Multi-Probe, FARGO incorporates an accurate and efficient global multi-probing strategy (GMP) that generates a global probing sequence from all L hash tables so that we enable to examine the candidates from “more promising" hash buckets across all hash tables in prior to “less promising" ones. To improve the quality of candidates in the global probing sequence, we exploit the strong relationship between the inner product and a so-called quantization distance between the query point and a candidate point.

In addition, we develop two optimization techniques to further improve the performance. First, we develop a novel asymmetric transformation, called random XBOX-Transformation (RXT), that maps the points in two different directions randomly so that

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

Table 1: Summary of Notations

Notation	Definition
$\mathcal{D}$	Dataset of points in $\mathbb{R}^d$
$n = \lvert\mathcal{D}\rvert$	Dataset cardinality
d	Dimensionality
x	A point in $\mathcal{D}$
$h(x)$	Hash functions
q	The query point
$q^\top x$	The inner product between q and x

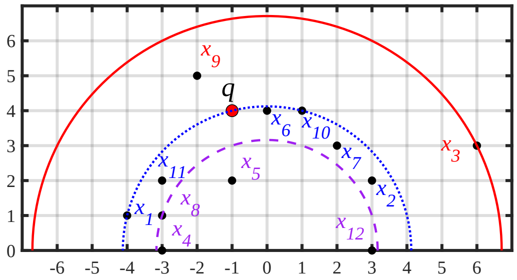


Figure 1: A Running Example

the points are distributed more evenly. There exist three main asymmetric transformation methods: (1) the L2-Transformation [38], (2) the Correlation-Transformation [39], and (3) the XBOX-Transformation [5, 45]. However, each of these transformations suffers from shortcomings that impact their performance negatively. The L2-Transformation and Correlation-Transformation introduce distortion errors. The XBOX-Transformation avoids distortion errors, but brings data distribution imbalances to the transformed dataset. So we develop RXT that eliminates these issues. Second, we develop a novel and training-free adaptive early termination strategy that enables a better trade-off between accuracy and efficiency. Although existing studies adopt an adaptive early termination condition for the approximate NNS problem [15, 26, 29, 30] to trade-off between accuracy and efficiency, they are either time-consuming on training or not well-adapted for solving the MIPS problem.

The major contributions of FARGO are as follows:

- We propose an efficient and accurate method called GMP to answer MIPS queries that exploits the properties of the inner product to examine high quality candidates.
- We develop two optimization techniques to further improve the performance. One is a novel asymmetric transformation method RXT that reduces data distribution imbalance. The other is a global adaptive early termination that determines its termination conditions adaptively.
- We conduct an extensive performance study using real datasets that covers the state-of-the-art methods for MIPS, finding that FARGO is efficient as well as accurate in terms of both the overall ratio and recall.

The rest of the paper is organized as follows. Section 2 presents the preliminaries. Section 3 introduces our solution FARGO. Section 4 develops two optimization techniques. Section 5 provides the theoretical analysis. Section 6 covers experimental studies that offer insight into the performance of the proposed FARGO and the main competitors. Section 7 reviews related work. Finally, Section 8 concludes the paper.

2 PRELIMINARIES

In this section, we introduce the preliminary knowledge. Frequently used notation is summarized in Table 1.

2.1 Problem Definition

Definition 1 (Maximum Inner Product Search). *Given a dataset $\mathcal{D} \subseteq \mathbb{R}^d$ that contains n points* (a.k.a. *vectors) and a query point $q \in \mathbb{R}^d$, the maximum inner product search (MIPS) returns a point $p \in \mathcal{D}$ that has the maximum inner product with q,* i.e.,

$$p = \arg\max_{x \in \mathcal{D}} q^\top x. \tag{1}$$

The MIPS problem can be transformed to the nearest neighbor search (NNS) problem that aims to find a point x closest (most similar) to q. For example, the NNS problem in terms of Cosine similarity is as follows:

$$p = \arg\max_{x \in \mathcal{D}} \frac{q^\top x}{\|q\|\|x\|} = \arg\max_{x \in \mathcal{D}} \frac{q^\top x}{\|x\|}. \tag{2}$$

It is well known that finding the exact NN in high-dimensional space is inherently computational expensive due to the 'curse of dimensionality'. A more attractive and generally sufficient alternative is to find an approximate nearest neighbor. In this paper, we focus on the transformation that converts an approximate maximum inner product search (c-MIPS) to an approximate nearest neighbor search (c-ANNS). We define them formally as follows:

Definition 2 (c-Approximate Nearest Neighbor Search). *Given a query point $q \in \mathbb{R}^d$ and an approximation ratio $0 < c < 1$, let $sim(\cdot)$ be the similarity metric and $x^* \in \mathcal{D}$ be the exact nearest neighbor of q. A c-approximate nearest neighbor search (c-ANNS) returns a point $x \in \mathcal{D}$ satisfying $sim(x, q) \geq c \cdot sim(x^*, q)$.*

Definition 3 (c-Maximum Inner Product Search). *Given a query point $q \in \mathbb{R}^d$ and an approximation ratio $0 < c < 1$, let $x^* \in \mathcal{D}$ be the point that has the maximum inner product with q. A c-maximum inner product search (c-MIPS) returns a point $x \in \mathcal{D}$ satisfying $q^\top x \geq c \cdot q^\top x^*$.*

Without confusion, we use MIPS and c-MIPS interchangeably in the following.

Example 1. *Fig. 1 shows a query point q and a dataset with 12 points. x_9 is the exact MIPS of q with $q^\top x_9 = 22$. When $c = 0.5$, the c-MIPS returns any point in $\{x_6, x_9, x_{10}, x_{11}\}$ because the inner product is larger than $0.5 \times 22 = 11$.*

2.2 Asymmetric Transformation

Given that the definitions of c-MIPS and c-ANNS are similar, the idea of considering the inner product as a similarity metric and then applying locality sensitive hashing (LSH), one of the most popular tools for c-ANNS, to solve c-MIPS is natural. However, a study [38] shows that the classical LSH framework is restrictive when solving MIPS. The existing proposals adopt an asymmetric transformation that maps the data points from the original space to a transformed space, then apply LSH to return the c-ANN in the transformed space as the results of the c-MIPS.

For better understanding, we briefly introduce an existing asymmetric transformation method, called XBOX-Transformation (XT) [19, 23, 33, 45], that consists of a function $P(x)$ for data point x and a function $Q(q)$ for query point q.

$$P(x) = [x; \sqrt{M^2 - \|x\|^2}], \tag{3}$$

$$Q(q) = [q; 0], \tag{4}$$

where [;] is the concatenation and $M = \max_x \|x\|$. Then, the Cosine similarity and Euclidean distance between $P(x)$ and $Q(q)$ can be computed as follows:

$$\text{sim}(P(x), Q(q)) = \frac{q^\top x}{\|q\| \cdot M}, \tag{5}$$

$$\text{dist}(P(x), Q(q)) = \|P(x), Q(q)\|^2 = \|q\|^2 + M^2 - 2q^\top x. \tag{6}$$

Unlike the other transformation methods, such as L2-Transformation [38] and Correlation-Transformation [39] whose distances in the transformed space are approximations, XBOX-Transformation is an exact transformation, since both the Cosine similarity and Euclidean distance between $P(x)$ and $Q(q)$ can be computed as the inner product between x and q without distortion.

2.3 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is a family of hash functions that enables c-ANNS in sublinear time with constant probability. An important property of LSH is that points with larger similarity have higher collision probability on their hash values.

Definition 4 (Locality Sensitive Hashing). *Given a similarity S_0, an approximation ratio $0 < c < 1$, probability values p_1 and p_2, where $p_1 > p_2$, a family $\mathcal{H} = \{h : \mathbb{R}^d \longrightarrow U\}$ is called (S_0, cS_0, p_1, p_2)-locality sensitive, if for any $x, y \in \mathbb{R}^d$, it satisfies the following two conditions:*

(1) If $sim(x, y) \geq S_0$ then $Pr[h(x) = h(y)] \geq p_1$
(2) If $sim(x, y) \leq cS_0$ then $Pr[h(x) = h(y)] \leq p_2$

LSH is a generic framework and two LSH families for different similarity metrics are adopted widely.

(1) One study [7] proposes an LSH family, called signed random projection (SRP), for Cosine similarity, which is defined as follows.

$$h(x) = sign(a^\top x), \tag{7}$$

where a is a random vector with each dimension $a_i \sim N(0, 1)$.

(2) Another study [10] proposes an LSH family, called fixed window w random projection (w-RP), for Euclidean distance, which is defined as follows.

$$h(x) = \lfloor \frac{a^\top x + b}{w} \rfloor, \tag{8}$$

where a is a random vector with each dimension $a_i \sim N(0, 1)$, b is a real number uniformly drawn from $[0, w)$, and w is a user-specified window size.

After the transformation, a c-MIPS query is converted to a c'-ANNS query that can be solved using existing LSH methods. Note that $c \neq c'$ and the approximation ratio c' is a key factor that determines the difficulty of answering the c'-ANNS query when using the (K, L)-bucketing strategy [14, 16, 20], which has complexity $O(n^{1/c'})$. Moreover, c' differs for similarity metrics, and a small c' incurs a high complexity.

Assume that $\|q\| = M$ and x^* is q's exact MIPS point, so $P(x^*)$ is $Q(q)$'s exact NNS point in the transformed space. Since $q^\top x^* < M^2$, we assume $q^\top x^* = l \cdot M^2$ and $0 < l < 1$. The c-MIPS query returns a point x where $P(x)$ is the result of the c'-ANNS query, and we have $q^\top x = c \cdot l \cdot M^2$. Let $c_1 = c'$ be the approximate ratio of x for Cosine similarity and $c_2 = c'$ be that of x for Euclidean distance. We demonstrate that c_1 is always larger than c_2 as follows. According to Eq. 3, we know that $\|P(x)\| = \|P(x^*)\| = M$.

For Cosine similarity, we have:

$$c_1 = \frac{\theta_{P(x),Q(q)}}{\theta_{P(x^*),Q(q)}} = \frac{\arccos \frac{q^\top x}{M \cdot M}}{\arccos \frac{q^\top x^*}{M \cdot M}} = \frac{\arccos (c \cdot l)}{\arccos (l)}.$$

For Euclidean distance, we have:

$$c_2 = \frac{\|P(x), Q(q)\|}{\|P(x^*), Q(q)\|} = \frac{\sqrt{2M^2 - 2c \cdot l \cdot M^2}}{\sqrt{2M^2 - 2 \cdot l \cdot M^2}} = \frac{\sqrt{1 - c \cdot l}}{\sqrt{1 - l}}.$$

It holds that $c_1 > c_2$ for any $c, l \in (0, 1)$. Therefore, we choose Cosine similarity as the metric and SRP as the LSH family for the c-ANNS in the transformed space, which incurs a smaller complexity.

3 OUR SOLUTION

We proceed to introduce a fast search framework FARGO for MIPS on large-scale, high-dimensional data. First, we present the data preprocessing and indexing. For the data preprocessing, we adopt a norm ranging strategy [19, 45] that divides the data points into s disjoint partitions based on the norm, and processes them in descending order of the maximum norm. For index construction, we employ the (K, L)-bucketing strategy [3] that builds hash tables for each partition. Second, we develop global multi-probing (GMP) for computing c-MIPS queries. Unlike Multi-Probe that maintains a local probing sequence for each hash table and examine them one by one, we maintain only a global probing sequence for all hash tables so that improves the query efficiency.

3.1 Data Preprocessing and Indexing

3.1.1 ***Data Preprocessing.*** Intuitively, a large norm $\|x\|$ leads to a large inner product $q^\top x$ with high probability, since $q^\top x$ is directly proportional to the norm $\|x\|$. Existing studies use so-called norm ranging to divide the data points into disjoint partitions, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_s$, according to their norms. We follow a strategy similar to that of an existing study [19] that uses two parameters b_0 and N_0 to bound the range of norms, where b_0 is the norm ratio, and N_0 is the maximum number of points in each partition. The partitions satisfy three conditions:

(1) For any two points $x \in \mathcal{D}_i$ and $y \in \mathcal{D}_j$, if $i < j$, we have $\|x\| \leq \|y\|$.
(2) For any two points $x, y \in \mathcal{D}_i$, if $\|x\| \leq \|y\|$, we have $b_0 \cdot \|y\| < \|x\|$, $b_0 \in (0, 1)$.
(3) For each partition $\mathcal{D}_i$, we have $|\mathcal{D}_i| < N_0$.

Example 2. *Assuming $N_0 = 6$ and $b_0 = 0.7$, the dataset in Fig. 1 is divided into three partitions, as indicated by three different colors.*

After partitioning the data, we apply XBOX-Transformation in each partition $\mathcal{D}_i$, so we have $P(x) = [x; \sqrt{M_i^2 - \|x\|^2}]$, where $M_i = \max_{x \in \mathcal{D}_i} \|x\|$ is the maximum norm of points in $\mathcal{D}_i$. Given a query point q, the maximum inner products between points in $\mathcal{D}_i$ and q are smaller than $M_i\|q\|$. We conduct ANNS from partitions $\mathcal{D}_s$ to $\mathcal{D}_1$, i.e., the partition with a larger norm will be examined earlier. Before examining a partition $\mathcal{D}_i$, we estimate whether we can find a better result with the condition $q^\top x_0 \geq cM_i\|q\|$, where x_0 is the best-found result. If the condition is satisfied, there does not exist a point in the remaining partitions whose inner product to q times c is greater than $q^\top x_0$, so we can safely terminate the query processing.

3.1.2 *Building Hash Tables.*

As mentioned, for each data point $P(x)$ in the transformed space, we adopt SRP as the LSH family so that $h(P(x)) = sign(a^\top P(x))$. For simplicity, we denote $h(P(x))$ by $h(x)$. Let $\theta_{P(x),Q(q)}$ be the angular distance between $P(x)$ and $Q(q)$, and we have

$$\theta_{P(x),Q(q)} = \arccos \frac{Q(q)^\top P(x)}{\|Q(q)\| \cdot \|P(x)\|} = \arccos \frac{q^\top x}{\|q\| \cdot M}. \tag{9}$$

We use the angular distance and the Cosine similarity interchangeably when this does not cause ambiguity. According to the property of SRP, we have the following lemma.

Lemma 1. *For a query point q and a data point x, let θ be the angular distance between $Q(q)$ and $P(x)$ in the transformed space. Without loss of generality, we assume that $\|q\| = 1$. From Eq. 9, since M is a constant, the inner product of x and q is inversely proportional to the angular distance between $Q(q)$ and $P(x)$. The collision probability of $h(x)$ and $h(q)$ is computed as follows [33].*

$$\Pr[h(x) = h(q)] = 1 - \frac{\theta}{\pi} \tag{10}$$

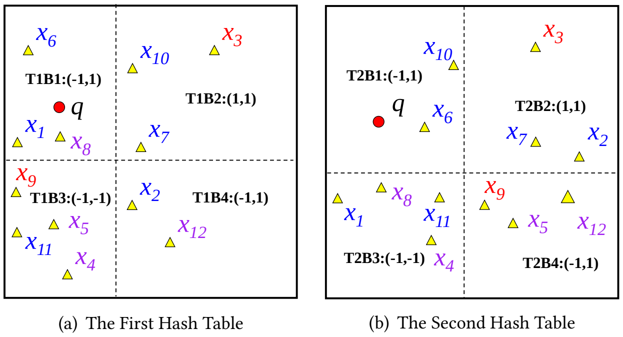
Next, we build hash tables using the (K, L)-bucketing strategy. After the norm ranging, for each partition $\mathcal{D}_i$, we build L hash tables and choose K hash functions $\mathcal{H} = \{h_1, \cdots, h_K\}$ for each hash table. This yields $K \cdot L$ hash functions and $2^K L$ hash buckets for each partition. To simplify the computation and reduce the number of hash functions, all s partitions use the same $K \cdot L$ hash functions to build the index.

3.2 Global Multi-Probing

We proceed to introduce our query processing algorithm, GMP, for computing c-MIPS queries. To examine candidate points, we use a global multi-probing strategy to generate a probing sequence. Specifically, we evaluate the probability that a hash bucket B contains the nearest neighbor via a so-called quantization distance (QD). The smaller the QD is, the higher the probability that B contains the nearest neighbor is. We determine which bucket to probe next based on the global probing sequence.

3.2.1 *Overview of Global Multi-Probing.*

Unlike Multi-Probe [31] that maintains a local probing sequence for each hash table, GMP generates a global probing sequence from all L hash tables. We illustrate the advantage of GMP over Multi-Probe with the following example.

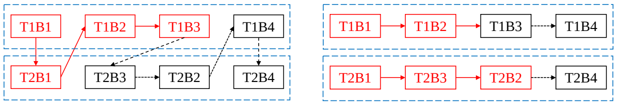


(a) The First Hash Table (b) The Second Hash Table

Figure 2: Point Projection on Two Hash Tables

Buckets	T1B1	T1B2	T1B3	T1B4	T2B1	T1B2	T2B3	T2B4
QD	0	2.7	5.8	8.5	0	9.0	6.6	15.6

(a) QDs of buckets



(b) GMP (c) Multi-Probe

Figure 3: Probing Sequences of GMP and Multi-Probe

Example 3. *Fig. 2 shows two hash tables after the projection by using SRP. Assume that for the first hash table, we have $a_{1,1} = (1.0, 0.1, -0.8)$, $a_{2,1} = (0.2, 0.9, -1.3)$. For the second hash table, we have $a_{1,2} = (0.6, -0.6, 1.1)$ and $a_{2,2} = (0.4, 0.9, -1.2)$. The projected values of q are $(\zeta_{1,1}(q), \zeta_{2,1}(q)) = (-1.6, 2.4)$ and $(\zeta_{1,2}(q), \zeta_{2,2}(q)) = (-3.0, 2.6)$. We denote by TiBj the j-th bucket in the i-th hash table. The QD between q and each hash bucket is shown in Fig. 3(a) (The computation of QD is introduced in Sec. 3.2.2). By examining the same number of candidate points (excluding the repeated points), say 8 points, we compare the accuracy of GMP and Multi-Probe. In Multi-Probe, we examine 4 candidate points in each hash table based on the local probing sequence in 3(c). In GMP, we examine 8 candidate points based on the global probing sequence in 3(b). We indicate the examined buckets with red color. In GMP, we examine T1B1, T2B1, T1B2, and T1B3, and we return x_9. In Multi-Probe, we examine T1B1, T1B2, T2B1, T2B3, and T2B2, and we return x_6. Note that, x_9 is a better result than x_6, which shows that GMP achieves a higher accuracy than Multi-Probe when examining a same number of points.*

In the query phase, we first compute $Q(q)$ based on Eq. 15 and then project $Q(q)$ with the K hash functions for each hash table. Since the dataset is divided into s partitions, we examine them in descending order of the maximum norm. For each partition $\mathcal{D}_i$, we determine which bucket to probe next among all $L \cdot 2^K$ buckets based on the global probing sequence. This process ends when a termination condition is satisfied. We address three important aspects of the GMP:

(1) Why is QD an effective indicator for probing hash buckets in all hash tables? (see Sec. 3.2.2)

(2) How to efficiently find the next hash bucket to probe among all unseen hash buckets? (see Sec. 3.2.3)
(3) How to set a proper termination condition to balance the accuracy and the efficiency? (see Secs. 3.2.5 and 4.2)

3.2.2 ***Quantization Distance.*** We proceed to show that the quantization distance (QD) is an effective indicator for probing, where a hash bucket B has a high QD is supposed to have high probability that contains the nearest neighbor. The key point is that QD is legal for ranking buckets from not only the same hash table, but also from different hash tables, which makes it possible to generate an effective global probing sequence.

Let $\zeta(x)$ be the projected vector of $P(x)$ under K hash functions, where $h_i(x) = sign(\zeta_i(x))$ and $\zeta_i(x) = a_i^\top P(x)$, $1 \le i \le K$. The quantization distance is defined as follows.

Definition 5 (Quantization Distance). *For a query point q and a bucket $B = (b_1, b_2, \cdots, b_K)$, the quantization distance between them is computed as follows.*

$$\mathrm{QD}(q, B) = \sum_{h_i(q) \neq b_i} |\zeta_i(q)|^2, 1 \le i \le K \qquad (11)$$

$\mathrm{QD}(q, B)$ is a random variable because it is the sum of a set of random variables $\zeta_i(q)$. Let $B(x)$ be the hash bucket of a data point x, we use the cumulative distribution function $\Pr[\mathrm{QD}(q, B(x)) \le \omega]$ to describe the distribution of $\mathrm{QD}(q, B(x))$. Note that QD between q and $B(x)$ is computed by using the same K hash functions in the same hash table. To show that QD is legal for ranking buckets from different hash tables, we first establish that $\Pr[\mathrm{QD}(q, B(x)) \le \omega]$ is only dependent on q and x, and is not dependent on hash functions in different hash tables. For simplicity, we denote the cumulative distribution function $\Pr[\mathrm{QD}(q, B(x)) \le \omega]$ of QD by $\varphi(\omega; \theta)$.

Lemma 2. *Given a data point x with the angle $\theta_{P(x),Q(q)} = \theta$ between q and x in the transformed space, the cumulative distribution function of $\mathrm{QD}(q, B(x))$ is determined only by θ and is independent on the hash table of $B(x)$.*

Lemma 3. *$\varphi(\omega; \theta)$ monotonically decreases with θ. In other words, for $0 \le \theta_1 < \theta_2 \le \pi$ and $\omega > 0$, we have $\varphi(\omega; \theta_1) > \varphi(\omega; \theta_2)$.*

Lemma 3 indicates that the smaller $\theta_{P(x),Q(q)}$ is, the smaller $\mathrm{QD}(q, B(x))$ is likely to be. Lemmas 2 and 3 reveal that the quantization distance is an effective indicator for global probing in FARGO. The proofs of Lemmas 2 and 3 are provided in Section 5.1.

3.2.3 ***Generating Global Probing Sequence.*** Obviously, it is expensive to compute the QDs of all $2^K \cdot L$ buckets to q and sort them to construct a complete global probing sequence. Fortunately, by maintaining a small-sized heap, we are able to obtain efficiently the bucket with the least QD among all unseen buckets.

Let $\zeta_{i,j}(q)$ be the i-th entry of the projected vector in the j-th hash table. For a bucket $B = (b_1, \cdots, b_K)$, from Eq. 11, we know that only the entries $\zeta_{i,j}(q)$ that have $h_{i,j}(q) \neq b_i$ contribute to QD between q and B. Therefore, we can represent B by a set $S^j(B)$ that only records the indexes of the entries that differ from those of $h_{i,j}(q)$.

$$S^j(B) = \{i \mid b_i \neq h_{i,j}(q), 1 \le i \le K\}$$

We reformulate Eq. 11 as follows.

$$\mathrm{QD}(q, B) = \sum |\zeta_{i,j}(q)|^2, i \in S^j(B) \qquad (12)$$

For simplicity, given the projected vector of q, we assume that $|\zeta_{i,j}(q)| < |\zeta_{i',j}(q)|$ for any $i < i'$. If not, we store their ascend order. As a result, the smaller the entries in $S^j(B)$ are or the fewer entries $S^j(B)$ has, the smaller $\mathrm{QD}(q, B)$ is. Note that by recording the indexes of the corresponding entries in the original projected vector of q, it is easy to reconstruct the hash bucket for a given set. Therefore, we proceed to generate a sequence of sets.

In the probing phase, we first probe the L buckets, where q is located in the L hash tables. We maintain a min-heap H that stores the candidate sets in ascending order of QD. Initially, we insert the set $S^j(B) = \{1\}$ into H for each hash table, since the hash bucket B that corresponds to $S^j(B) = \{1\}$ has the smallest QD to q in the j-th hash table. Next, we continue to process the top set of H and insert two sets into H that are generated by operations Shift() and Expand().

Let i_{max} be the maximum entry of the current top set $S^j(B)$.

(1) Shift($S^j(B)$) replaces i_{max} by $i_{max} + 1$ in $S^j(B)$ and returns $S^j(B')$, which ensures that $S^j(B')$ is the set to generate that contains $S^j(B)/\{i_{max}\}$ and that has the smallest QD.
(2) Expand($S^j(B)$) inserts the entry $i_{max} + 1$ into $S^j(B)$ and returns $S^j(B')$, which ensures that $S^j(B')$ is the set to generate that contains $S^j(B)$ with the smallest QD.

When $i_{max} = K$, no new set is inserted into H. The operations Shift() and Expand() ensure the correctness of the generated probing sequence. We use an example to explain how Shift() and Expand() operations work. Consider that $L = 2$, $K = 4$, and $h_1(q) = (-0.2, 0.4, 0.5, -0.6)$, q falls into the bucket B_0=(-1,1,1,-1) and $S^1(B_0) = \varnothing$. B_1=(1,1,1,-1) satisfies $S^1(B_1) = \{1\}$. For B_2=(-1,-1,-1,-1) that $S^1(B_2) = \{2, 3\}$. $S^1(B_S)$=Shift($S^1(B_2)$) $= \{2, 4\}$ and $S^1(B_E)$=Expand($S^1(B_2)$) $= \{2, 3, 4\}$. Since $i_{max} = K = 4$ in $S^1(B_S)$ and $S^1(B_E)$, we do not conduct the Shift() and Expand() operations on them. Based on the definition of $S^1()$, B_S and B_E are buckets (-1,-1,1,1) and (-1,-1,-1,1), respectively.

3.2.4 ***From c-MIPS to (I, c)-MIPS.*** After the transformation, a c-MIPS query is converted to a c'-ANNS query. An LSH method cannot solve a c'-ANNS problem directly, but can solve it by conducting a sequence of threshold-based (θ, c')-ANNS queries for angular distance by increasing the angle θ. Therefore, to answer a c-MIPS query, we need to execute a sequence of (I, c)-MIPS queries as defined below while decreasing the inner product I.

Definition 6 ((I, c)-MIPS). *Assume a query point q, a dataset $\mathcal{D}$, an inner product threshold I, and an approximation factor $c < 1$. The (I, c)-MIPS query returns the following result:*

(1) If at least one point x exists in $\mathcal{D}$ such that $q^\top x > I$, it returns a point x' in $\mathcal{D}$ such that $q^\top x' > c \cdot I$;
(2) If no point x exists in $\mathcal{D}$ such that $q^\top x > c \cdot I$, it returns nothing.

Specifically, we set an initial value I_{max} and keep compressing I with c in multiple rounds, i.e., $I = I_{max}, cI_{max}, c^2 I_{max}, \ldots$, until the termination condition is satisfied, where I_{max} is the maximum possible inner product. Each (I, c)-MIPS query corresponds to a (θ, c')-ANNS query as defined below that can be answered via LSH.

Definition 7 ((θ, c')-ANNS). *Assume a query point q, a dataset $\mathcal{D}$, an angle threshold θ, and an approximation factor $c' > 1$. The (θ, c')-ANNS query returns the following result:*

(1) *If at least one point x exists in $\mathcal{D}$ such that $\theta_{q,x} < \theta$, it returns a point x' in $\mathcal{D}$ such that $\theta_{q,x'} < c' \cdot \theta$;*
(2) *If no a point x exists in $\mathcal{D}$ such that $\theta_{q,x} < c' \cdot \theta$, it returns an empty set.*

To ensure that the (I, c)-MIPS query is correctly answered via the (θ, c')-ANNS query, we consider two points x_1 and x_2 where $q^\top x_1 = I$ and $q^\top x_2 = cI$. If x_1 is the exact MIP result, x_2 would be a correct (I, c)-MIPS result. Therefore, we can set θ and c' to make $P(x_1)$ be the exact NNS result and $P(x_2)$ be the correct (θ, c')-ANNS result. That is,

$$\begin{cases} \theta = \langle Q(q), P(x_1) \rangle = \arccos(\frac{I}{M \cdot \|q\|}) \\ c'\theta = \langle Q(q), P(x_2) \rangle = \arccos(\frac{cI}{M \cdot \|q\|}) \end{cases}$$

By solving the equations, we have:

$$\theta = \arccos(\frac{I}{M \cdot \|q\|}), \; c' = \frac{\arccos(\frac{c \cdot I}{M \cdot \|q\|})}{\arccos(\frac{I}{M \cdot \|q\|})} \tag{13}$$

3.2.5 ***Termination Condition.*** Let T be the maximum number of points to probe and let t be a QD threshold. For a (θ, c')-ANNS query, the process terminates when one of the following conditions is satisfied:

(1) We verify T candidate points;
(2) With B^* being the bucket to probe next, $\mathrm{QD}(q, B^*) > t$.

To ensure that FARGO returns correct (θ, c')-ANNS result, we determine the values of T and t as follows.

$$\begin{cases} (1 - \varphi(t; \theta))^L = 1/e \\ T = 2nL \cdot \varphi(t; c'\theta) \end{cases} \tag{14}$$

The intuition of Eq. 14 is that when the first condition is satisfied, we have found a correct (θ, c')-ANNS result with at least probability $1 - 1/e$; and when the second condition is satisfied, since there is at most $n \cdot \varphi(t; c'\theta)$ false positive points in a hash table, by using Markov's inequality, we have examined all false positive points in L hash tables with probability at least $1/2$. That is,

$$1 - \frac{1 - (1 - \varphi(t; c'\theta))^L}{2L \cdot \varphi(t; c'\theta)} > 1 - \frac{L \cdot \varphi(t; c'\theta)}{2L \cdot \varphi(t; c'\theta)} = \frac{1}{2}.$$

We call such a termination condition as the normal termination condition (NT). NT is designed in a similar way as many mainstream LSH methods [10, 14, 42, 46], which guarantees to answer a c^2-MIPS problem with a constant probability $1/2 - 1/e$ [14]. As mentioned, we answer a c^2-MIPS query by conducting multiple (I, c)-MIPS queries. With x^* being q's exact MIPS point with $q^\top x^* = I_{max}$, we set $I = I_{max}$ for the first (I, c)-MIPS query. For each (I, c)-MIPS query, we convert the (I, c)-MIPS query to a (θ, c')-ANNS query based on Eq. 13 and use NT to terminate the probing process. Once we find a point x with $q^\top x \geq cI$, we return it as the result. Otherwise, we decrease I to cI and conduct a (cI, c)-MIPS query.

4 OPTIMIZATIONS

We proceed to introduce two optimization techniques to further improve the performance. First, we propose a novel asymmetric transformation method, called random XBOX-Transformation (RXT), that reduces data distribution imbalance. Second, we develop a global adaptive early termination (AET) that determines its termination conditions adaptively.

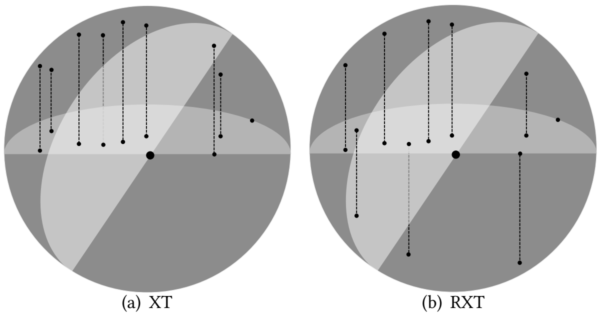

(a) XT
(b) RXT

Figure 4: Comparison of RXT with XT

4.1 Random XBOX-Transformation

We develop a novel random XBOX-Transformation (RXT) to convert a c-MIPS query into a c'-ANNS query, which eliminates distortion errors and reduces data distribution imbalances when compared with existing asymmetric transformations.

Fig. 4(a) shows an example of XT. The horizontal half-pane is the original space, and the points on it are the data points. After transformation, all points are mapped to the upper sphere with an identical length. The inclined half-pane is a hash function that partitions the points to two sides. We can see that most points locate in one side, and only a few points are in the other side. Therefore, this distribution imbalance may impact the performance negatively.

Similar to XT, our RXT has two functions $P(x) : \mathcal{R}^d \mapsto \mathcal{R}^{d+1}$ and $Q(q) : \mathcal{R}^d \mapsto \mathcal{R}^{d+1}$ that are defined as follows.

$$\begin{aligned} P(x) &= [x; r \cdot \sqrt{M^2 - \|x\|^2}] \\ Q(q) &= [q; 0], \end{aligned} \tag{15}$$

where $r \in \{1, -1\}$ is a random variable and $\Pr[r = 1] = \Pr[r = -1] = \frac{1}{2}$. Note that RXT still satisfies Eq. 5, so it is an exact transformation without distortion errors. The difference of RXT from XT is that the sign of the $(d + 1)$-st dimension of each transformed point $P(x)$ has 50% probability to be -1. Fig. 4(b) shows an example of RXT where the points are mapped to both the upper and lower spheres with equal probability, and the points are more evenly distributed between the two sides partitioned by the hash function.

We analyze the advantage of RXT over XT by considering the probabilities of $h(x) = 1$ and $h(x) = -1$.

Lemma 4. *Using XT, the number of points with hash value* 1 *is always different with the number of points with hash value* -1*; while using RXT, it is likely that the number of points with hash value* 1 *equals to the number of points with hash value* -1.

Proof. For XT, we have $h(x) = sign(a^\top P(x)) = sign(\sum_{i=1}^{d} x_i a_i + \sqrt{M^2 - \|x\|^2} \cdot a_{d+1})$. Let $A = \frac{\sum_{i=1}^{d} x_i a_i}{\|x\|}$, we have $A \sim N(0, 1)$ according to the property of the sum of the normal distribution since

$a_i \sim N(0,1)$. Thus, $h(x) = sign(\|x\|A + \sqrt{M^2 - \|x\|^2} \cdot a_{d+1})$.

$$\Pr[h(x) = -1] = \Pr[A < -\frac{\sqrt{M^2 - \|x\|^2} \cdot a_{d+1}}{\|x\|}]$$

Consider the conditional probability $\Pr[h(x) = -1 \mid a_{d+1} = u] = \phi(-\frac{\sqrt{M^2 - \|x\|^2} \cdot u}{\|x\|})$, where $\phi(\cdot)$ is the cumulative distribution function of the standard normal distribution. If $u > 0$, then $-\frac{\sqrt{M^2 - \|x\|^2} \cdot u}{\|x\|} < 0$ and $\Pr[h(x) = -1 \mid a_{d+1} = u] < 1/2$ for any point x. In this case, it is likely that more than $n/2$ points have hash value 1. Likewise, if $u < 0$, it is likely that more than $n/2$ points have hash value -1. For RXT, since $\Pr[r = 1] = \Pr[r = -1] = 1/2$, we have:

$$\Pr[h(x) = -1 | a_d + 1 = u] = \phi(-\frac{r\sqrt{M^2 - \|x\|^2} \cdot u}{\|x\|})$$
$$= \frac{1}{2}\left[\phi(-\frac{\sqrt{M^2 - \|x\|^2} \cdot u}{\|x\|}) + \phi(\frac{\sqrt{M^2 - \|x\|^2} \cdot u}{\|x\|})\right] = \frac{1}{2}.$$

No matter what a_{d+1} is, it is expected that $n/2$ points have hash value 1, which completes the proof. □

Lemma 4 implies that the points are more likely to be mapped to one side of the hyperplane using XT and that they are likely to be mapped evenly to the two sides of the hyperplane when using RXT. Although RXT is a slight modification from XT, it reduces the data distribution imbalance after the transformation, which benefits the subsequent point probing process. The experiments also suggest that RXT is an effective replacement of XT.

4.2 Global Adaptive Early Termination

NT employs two parameters T and t that are fixed for all queries. However, the number of points to probe to reach a given accuracy varies across query points, which lowers the performance when facing data skew. Motivated by this, we develop a global adaptive early termination (AET) strategy that "adaptively" takes both the best-found result and $QD(q, B)$ into consideration. Assume that B^* is the next bucket to probe and I_0 is the currently found maximum inner product. We define two events according to I_0 and $QD(q, B^*)$.

- **E1:** For a point x with $q^\top x \geq I_0/c$, the hash bucket in which x is located in a certain hash table remains unseen.
- **E2:** For a point x with $q^\top x \geq I_0/c$, in L hash buckets in which x is located, there is at least one that remains unseen.

When the probability that **E2** happens, $\Pr[E_2]$, is small enough, the probability that we find a better result in the remaining buckets is small. It is then reasonable to terminate the query process. So we design AET by setting a failure probability p_τ as follows:

$$\begin{cases} \alpha_1 = 1 - \varphi(QD(q, B^*); \arccos(\frac{I_0}{c \cdot M_i \cdot \|q\|})) \\ \alpha_2 = 1 - (1 - \alpha_1)^L < p_\tau \end{cases} \tag{16}$$

Here, α_1 equals $\Pr[E_1]$ and $\alpha_2 = 1 - (1 - \Pr[E_1])^L$ equals $\Pr[E_2]$. Based on the intermediate query results, $QD(q, B^*)$ and I_0, AET verifies whether $\Pr[E_2]$ is within the allowed failure probability p_τ. When AET is satisfied, this indicates that the found results in $\mathcal{D}_i$ are correct results of the c-MIPS query.

We describe how to compute a c-MIPS query using our solution that adopts AET as the termination condition in Algorithm 1. We conduct the c-MIPS query from partitions $\mathcal{D}_s$ to $\mathcal{D}_1$ (Line 3). In each partition D_i, we probe the buckets in ascending order of their QD to q (Line 8). The approach of generating the probing sequence described in Sec. 3.2.3 is used to determine the next buckets to be examined. If AET is satisfied after examining the bucket, we terminate the query in D_i (Line 7). From Algorithm 1, AET has the following advantages compared to NT: (1) the establishment of AET depends not on a fixed, predefined threshold T and t, but on the intermediate query results, which eliminates many unnecessary candidates, especially when the data is skewed. (2) AET is simpler and more efficient since we do not need to evaluate an initial I_{max} and conduct multiple (θ, c')-ANNS queries.

Algorithm 1: Global Multi-Probing (GMP)

```
Input: A query point $q$, partitions $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_s$, $c$, $p_\tau$,
       and $L$ hash tables
Output: A point
1  Compute the hash values of $q$;
2  Let $x_0$ be the best-found MIPS result;
3  for $i = s \rightarrow 1$ do
4      $M_i \leftarrow \max_{x \in \mathcal{D}_i} \|x\|$;
5      if $q^\top x_0 \geq cM_i\|q\|$ then
6          Break;
7      while AET is not satisfied do
8          $B^* \leftarrow$ the bucket with the smallest QD in all the
           unseen buckets;
9          Verify the points in $B^*$ and update $x_0$;
10 return $x_0$;
```

GMP can be adapted easily to return k points with maximum inner products. We call the c-MIPS with k results (c, k)-MIPS. Specifically, to answer a (c, k)-MIPS query, we only need to set x_0 in Algorithm 1 to be the best-found k-th MIPS result. We give an example of computing a $(0.5, 2)$-MIPS query next.

Example 4. *In Fig. 1, we have* $\mathcal{D}_1 = \{x_4, x_5, x_8, x_{12}\}$, $\mathcal{D}_2 = \{x_1, x_2, x_6, x_7, x_{10}, x_{11}\}$, *and* $\mathcal{D}_3 = \{x_3, x_9\}$. *When a query q comes, we consider $\mathcal{D}_3$ first. We probe the hash tables according to the global probing sequence in Fig. 3(b). In this process, x_3 is found in T1B2, and x_9 is found in T1B3. Next, we consider $\mathcal{D}_2$. AET is satisfied after probing x_6 in T1B2. Then, the 2nd best-found MIPS result is x_6 and $q^\top x_6 = 16 > 0.5\|q\| \cdot M_1 = \sqrt{85/2}$, we terminate the query process and return x_9 and x_6.*

Remark. AET is simpler yet more efficient than existing adaptive termination strategies by only considering the relationship between the projection distance and the current best-found result. Studies [15] and [26] train learning models to predict when the search terminates, which is time-consuming on training. In contrast, AET is training-free and designed directly according to the property of quantization distance, which is easier to use without a large preprocessing time. EI-LSH [30] develops an adaptive termination condition by exploiting a smaller search radius and exploring a tighter quality guarantee, but it is only applicable for QA-LSH [18] and is difficult to extend to other LSH methods.

5 THEORETICAL ANALYSIS

We proceed to provide a theoretical analysis. First, we present theoretical guarantees. Second, we derive the time and space complexity of the algorithm.

5.1 Theoretical Guarantees

We prove Lemmas 2 and 3 that reveal that the quantization distance is an effective indicator for probing.

5.1.1 *Proof of Lemma 2*.

Proof. Assume a query point q and a data point x whose angle with q is $\theta \in (0, \pi)$, since $\mathrm{QD}(q, B(x))$ is the sum of a set of random variables, to determine the distribution of $\mathrm{QD}(q, B(x))$, we first analyze the distribution of each random variable in $\mathrm{QD}(q, B(x))$. Let $h_{i,j}$ be the i-th hash function in the j-th hash table. We define each item in $\mathrm{QD}(q, B(x))$ as follows.

$$\mu_{i,j}(q, x) = \begin{cases} |\zeta_{i,j}(q)|^2, & h_{i,j}(q) \neq h_{i,j}(x) \\ 0, & h_{i,j}(q) = h_{i,j}(x) \end{cases}$$

According to the property of a normal distribution, the cumulative distribution function of $\mu_{i,j}(q, x)$, $\Pr[\mu_{i,j}(q, x) \leq \omega]$, is

$$\psi(\omega; \theta) = 1 - \frac{\theta}{\pi} + \int_0^{\sqrt{\omega}} 2\phi(-u \cdot \cot\theta) d\phi(u), \tag{17}$$

where $\phi(x)$ is the cumulative distribution function of the standard normal distribution. Then, the probability density function of $\mu_{i,j}(q, x)$ can be written as $g(\omega; \theta) = \frac{\partial \psi(\omega;\theta)}{\partial \omega}$. For any two different hash buckets in which x is located in two hash tables, since $\zeta_{i,j}(q)$ are independently and identically distributed, where $j \in \{1, 2\}$ identifies the table and $i \in [1, K]$, the distribution of $\mathrm{QD}(q, B(x))$ can be expressed as:

$$\Pr[\mathrm{QD}(q, B(x)) \leq \omega] = \Pr[\sum_{i=1}^{K} \mu_{i,j}(q, x) \leq \omega] = \int_0^{\omega} g^{K*}(u; \theta) du,$$

where $g^{K*}(u; \theta)$ is the K-fold convolution of $g(u; \theta)$ [43] on u. The distribution of $\mathrm{QD}(q, B(x))$ is independent on a specific hash table, which indicates that no matter which hash table $B(x)$ belongs to, $\mathrm{QD}(q, B(x))$ has the same distribution. □

5.1.2 *Proof of Lemma 3*.

Proof. Let $\varphi_{(m)}(\omega; \theta)$ be the cumulative distribution function of $\mathrm{QD}(q, B(x))$ when we have m hash functions in each hash table. Let *State*(m) be the statement that $\varphi_{(m)}(\omega; \theta)$ monotonically decreases with θ. We prove Lemma 3 by induction on m.

Base case: We show that the statement *State*(1) holds for the smallest positive integer number $m = 1$. Specially, when $m = 1$, $\varphi_{(1)}(\omega; \theta) = \Pr[\mu_{i,j}(q, x) \leq \omega]$. Since

$$\frac{\partial \psi(\omega; \theta)}{\partial \theta} = -\frac{1}{\pi} + \frac{1}{\pi}[1 - \exp(-\frac{\omega}{2\sin^2\theta})] < 0,$$

$\psi(\omega; \theta)$ monotonically decreases with θ, and *State*(m) is true.

Inductive step: We show that for any $m \geq 1$, if *State*(m) holds, then *State*$(m + 1)$ also holds. The probability density function of $\mu_{i,j}(q, x)$ can be written as $g(\omega; \theta) = \frac{\partial \psi(\omega;\theta)}{\partial \omega}$. According to the distribution of the sum of variables,

$$\varphi_{(m+1)}(\omega; \theta_1) = \int_0^{\omega} g^{(m+1)*}(u; \theta_1) du,$$

where $g^{m*}(u; \theta)$ is the m-fold convolution of $g(u; \theta)$ [43] on u.

$$\begin{aligned} \int_0^{\omega} g^{(m+1)*}(u; \theta_1) du &= \int_0^{\omega} g(u; \theta_1) * g^{m*}(u; \theta_1) du \\ &= \int_0^{\omega} \int_0^{u} g(t; \theta_1) g^{m*}(u - t; \theta_1) dt du \end{aligned}$$

Interchanging the order of the integrals, we get:

$$\begin{aligned} &\int_0^{\omega} \int_0^{u} g(t; \theta_1) g^{m*}(u - t; \theta_1) dt du \\ &= \int_0^{\omega} g(t; \theta_1) dt \int_t^{\omega} g^{m*}(u - t; \theta_1) du \\ &= \int_0^{\omega} g(t; \theta_1) dt \cdot \varphi_{(m)}(\omega - t; \theta_1) \\ &> \int_0^{\omega} g(t; \theta_1) \varphi_{(m)}(\omega - t; \theta_2) dt = \int_0^{\omega} g(u; \theta_1) * g^{m*}(u; \theta_2) du \end{aligned}$$

Similarly, we have:

$$\begin{aligned} &\int_0^{\omega} g(u; \theta_1) * g^{m*}(u; \theta_2) du \\ &> \int_0^{\omega} g(u; \theta_2) * g^{m*}(u; \theta_2) du = \varphi_{(m+1)}(\omega; \theta_2) \end{aligned}$$

This shows that the statement *State*$(m + 1)$ also holds true, establishing the inductive step.

Conclusion: Having shown the base case and the inductive step, statement *State*(m) holds for every positive integer m. □

5.2 Algorithm Analysis

Assume that the dataset $\mathcal{D}$ is divided into partitions $\mathcal{D}_1, \ldots, \mathcal{D}_s$. Given $n_i = |\mathcal{D}_i|$, we have $n_i = O(1)$, following Huang et al. [19]. Therefore, we set L and K as constants to compute MIPS on $\mathcal{D}_i$. To build the indexes in $\mathcal{D}_i$, we require computing $K \cdot L$ hash values for each point, which incurs $O(KLd)$ cost. Hence, the space consumption and indexing time of FARGO for $\mathcal{D}_i$ are $O(n_i)$ and $O(n_i d)$. Hence, FARGO has space cost $O(n)$ and indexing time $O(nd)$.

The query cost depends on how many false positive points we verify during the search. Assume that AET is satisfied and the query terminates at the bucket B^* and returns x with $q^\top x = I_0$, we will not access any bucket B that $\mathrm{QD}(q, B) > \mathrm{QD}(q, B^*)$. However, Multi-Probe cannot guarantee to skip accessing these buckets, which makes it always access more buckets than that with GMP. For instance, if B^* is in the second hash table and x is found in the third table, the bucket B in the first and second hash tables are both likely to be accessed even though $\mathrm{QD}(q, B) > \mathrm{QD}(q, B^*)$. Since x is not found when we probe the first two hash tables. This demonstrate the superiority of GMP over Multi-Probe. Next, we analysis the query cost of FARGO. We denote by β the probability that a false positive point is verified, $\beta = 1 - (1 - \alpha)^L$ where $\alpha = \varphi(\mathrm{QD}(q, B^*); \arccos(\frac{I_0}{M_i \cdot \|q\|}))$. Then, the point accessed is at most $O(\beta n)$. Although it is nearly impossible to simplify β due to the complex formation of φ, we can easily find that $\beta < 1 - p_\tau$ since $\alpha < \alpha_1$. Moreover, $\varphi(t; \theta)$ decreases

Table 2: Datasets

Dataset	n ($\times 10^3$)	d	Dataset	n ($\times 10^3$)	d
Audio	54	192	YahooMusic	625	300
MNIST	60	784	GIST	1,000	960
Cifar	60	1,024	Tiny5M	5,000	384
Trevi	100	4,096	Tiny80M	79,302	384

rapidly with θ and $\arccos(\frac{I_0}{M_i \cdot \|q\|})$ is always much smaller than $\arccos(\frac{I_0}{c \cdot M_i \cdot \|q\|})$. Thus, β is usually much greater than $1 - p_\tau$. Given $c = 0.5$, $M_i = \|q\| = 1.0$, and $I_0 = 0.48$, by setting $p_\tau = 0.1$, we have $\beta \approx 0.06$.

Theorem 1. *FARGO has space cost $O(n)$, indexing time $O(nd)$ and query cost $O(\beta nd)$, where β is much smaller than 1.*

Proof. Since L and K are constants, the space consumption is $O(n)$. The query time cost comes from two parts: 1) generating probing sequences; and 2) verifying the inner products between candidate points and q. In each partition $\mathcal{D}_i$, the former has cost $O(n_i \log n_i)$, and the latter has cost $O(\beta n_i d)$. Note that the probing sequence is the same in all partitions since we use the same $K \cdot L$ hash functions to build the index. So, we only have to generate the probing sequence once. Therefore, the total query time is $O(n_i \log n_i + \sum_{i=1}^{s} \beta n_i d) = O(\beta nd)$. □

6 EXPERIMENTAL STUDY

We report on extensive experiments with real-world datasets that offer insight into the performance of FARGO. In particular, we aim to answer the following questions:

Q1: Why is RXT a better transformation method than XT in FARGO?

Q2: How does FARGO benefit from the adaptive early termination (AET) strategy?

Q3: Why is GMP a better probing sequence than Multi-Probe?

Q4: How does FARGO perform on the index size and indexing time relative to the competitors?

Q5: How does FARGO perform on the query processing relative to the competitors under default settings?

Q6: How does FARGO compare with the competitors in the Recall-Time dimensions and the Ratio-Time dimensions?

6.1 Experimental Settings

All algorithms are implemented in C++ and are compiled with g++ using -Ofast optimization. All experiments are run on a Ubuntu Server with 4 Intel(R) Xeon(R) Gold 6248 CPUs and 1.5 TB RAM.

6.1.1 ***Datasets and Query Sets.*** We use 8 real-world datasets varying in types, cardinality and dimensionality as shown in Table 2. Most of them are used in our competitor algorithms. Specifically, *YahooMusic* are often used in recommendation systems [6, 12, 24]. The other datasets are used widely in similarity search [19, 27, 46].

For queries, we randomly select 200 points from each dataset and repeat each experiment 20 times. We vary the value of k in $\{1, 10, 20, \ldots, 100\}$ and set the default value to 50. We vary the value of c in $\{0.1, 0.2, \ldots, 0.9\}$ and set the default value to 0.8.

6.1.2 ***Competing Algorithms.*** We have 6 state-of-the-art competitors for solving the c-MIPS problem:

(1) **Simple-LSH** [33] uses the XBOX-Transformation and the (K, L)-bucketing strategy.
(2) **Range-LSH** [45] applies norm-ranging strategy to partition the dataset and then employs Simple-LSH [33] in each partition.
(3) **H2-ALSH** [19] uses the XBOX-Transformation and QALSH [18] after applying norm-ranging.
(4) **RPT** [23] uses the XBOX-Transformation and randomized partition trees.
(5) **ProMIPS** [40] projects high-dimensional data points to low-dimensional ones and accesses them by the ascending order of their distance to query point in the low-dimensional space.
(6) **CeoMIPS** [35] chooses a few projections associated with the extreme values of the query signature and yields a sublinear query cost with search recall guarantee. Among three algorithms provided in [35], we adopts *CoCEOs* as our competitor.

6.1.3 ***Parameter Settings.*** FARGO needs to consider the parameters b_0, N_0, L, K, and p_τ. We set $K = 12$, $L = 5$, $N_0 = 20480$, $b_0 = \sqrt{0.95}$ and $p_\tau = 0.1$ by default. For H2-ALSH, we set $c_0 = 2.0$ and $N_0 = 5000$. For ProMIPS, we are advised to set $p = 0.5$, $m = 10$, $k_p = 5$, $N_{key} = 40$ and $k_{sp} = 10$. For CeoMIPS, we set $D = 1024$, $s = 5$ and $B = n/10$. For the other three algorithms, we perform tuning to obtain parameter settings that achieve best performance.

6.1.4 ***Evaluation Metrics.*** We use three performance metrics: query time (ms), overall ratio, and recall, where the query time evaluates efficiency and the overall ratio and recall evaluate result quality. For a query q, we denote the result of a (c, k)-MIPS query by $R = \{x_1, x_2, \cdots, x_k\}$. Let $R^* = \{x_1^*, x_2^*, \cdots, x_k^*\}$ be the exact k MIPS points. The overall ratio and recall are computed as follows.

$$OverallRatio = \frac{1}{k} \sum\nolimits_{i=1}^{k} \frac{q^\top x_i}{q^\top x_i^*} \tag{18}$$

$$Recall = \frac{|R \cap R^*|}{|R^*|} \tag{19}$$

6.2 Self Evaluation

To evaluate the performance of FARGO, we first demonstrate that RXT and AET are better choices for the transformation method and termination condition than are XT and NT. Then, we compare the performance of FARGO with Multi-Probe.

6.2.1 ***Comparison of RXT and XT (Q1).*** To compare RXT and XT, we denote by FARGO-XT our solution with XT as the transformation method. For each dataset, we repeat the experiment 100 times and plot the running time of FARGO and FARGO-XT via box-plot by normalizing them with the average query time of FARGO. The results of all dataset are shown in Fig. 5. We observe that FARGO is faster and more stable than FARGO-XT because the query time of FARGO usually has a smaller average value and a standard deviation in the box-plot. Two observations explain this. (1) In each experiment, the data distribution after applying RXT is more even. (2) For the 100 experiments, FARGO generates similar data partitions. That is, the index structure changes only little across the experiments.

Table 3: Performance Overview

		FARGO	H2-ALSH	Simple-LSH	Range-LSH	RPT	ProMIPS	CeoMIPS
Audio	Query Time (ms)	1.160	4.520	9.319	3.080	9.022	1.960	1.197
	Recall	0.9860	0.9782	0.9148	0.9500	0.9382	0.5504	0.8654
	Overall Ratio	0.9994	0.9991	0.9919	0.9968	0.9945	0.9095	0.9887
MNIST	Query Time (ms)	3.079	5.519	17.76	7.720	9.399	7.560	2.426
	Recall	0.9324	0.8790	0.7410	0.8396	0.712	0.7804	0.2732
	Overall Ratio	0.9985	0.9967	0.9866	0.9928	0.9826	0.9889	0.9411
Cifar	Query Time (ms)	0.08	2.439	15.80	0.641	12.99	2.720	2.425
	Recall	0.9984	0.9982	0.9302	0.9860	0.9114	0.5498	0.9042
	Overall Ratio	1.0000	1.0000	0.9971	0.9996	0.9972	0.9618	0.9985
Trevi	Query Time (ms)	0.16	7.159	9.9985	3.280	41.95	18.32	-
	Recall	0.9996	0.9994	0.9552	0.9876	0.7406	0.0042	-
	Overall Ratio	1.0000	1.0000	0.9993	0.9997	0.9918	0.9055	-
YahooMusic	Query Time (ms)	9.8427	21.88	72.47	17.20	27.92	71.48	12.56
	Recall	0.9982	0.9958	0.492	0.9586	0.3794	0.3716	0.9936
	Overall Ratio	0.9998	0.9997	0.6208	0.9859	0.4868	0.6324	0.9989
GIST	Query Time (ms)	0.4421	3.880	4.520	2.760	6.719	174.7	43.35
	Recall	0.9998	0.9998	0.1954	0.7952	0.4822	0.97	0.5868
	Overall Ratio	1.0000	1.0000	0.8104	0.9837	0.9239	0.9985	0.9560
TinyImages5M	Query Time (ms)	0.5594	8.279	26.63	75.11	37.16	464.5	102.5
	Recall	0.9874	0.9498	0.2214	0.5650	0.3662	0.7258	0.3296
	Overall Ratio	0.9999	0.9994	0.9551	0.9871	0.9706	0.9931	0.9722
TinyImages80M	Query Time (ms)	4.439	252.95	65.28	156.9	262.3	9960	1982
	Recall	0.8936	0.8218	0.1348	0.3794	0.2144	0.7712	0.2206
	Overall Ratio	0.9989	0.9980	0.9440	0.9848	0.9648	0.9960	0.9759

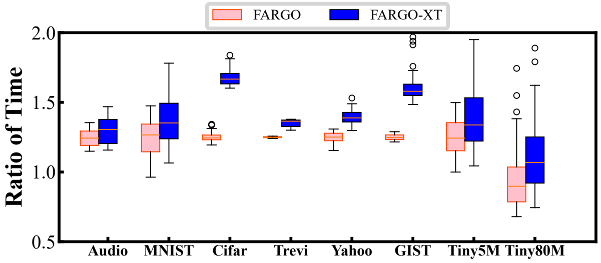
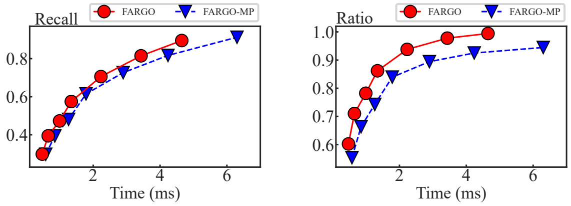
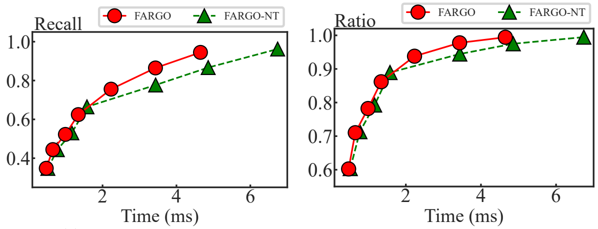


Figure 5: Comparison on RXT and XT



(a) Recall-Time Curve (b) Ratio-Time Curve

Figure 7: Comparison on GMP and Multi-Probe



(a) Recall-Time Curve (b) Ratio-Time Curve

Figure 6: Comparison on AET and NT

6.2.2 *Comparison of AET and NT (Q2).*

To compare the performance of AET and NT, we denote by FARGO-NT our solution with NT as the termination condition. We plot the recall-time and ratio-time curves on *YahooMusic* when varying p_τ for FARGO and varying T and t for FARGO-NT – see Fig. 6. We observe that FARGO takes less time than FARGO-NT to reach the same recall or overall ratio. Especially when the recall or overall ratio is higher, the time difference is larger. A key reason is that AET adaptively decides when to terminate so that fewer candidates are verified. Normally, to achieve a certain accuracy, the number of candidate points that need to be examined differs for each query point. NT does not consider this issue.

6.2.3 *Comparison of GMP and Multi-Probe (Q3).*

To demonstrate that GMP is a better probing sequence than Multi-Probe, we denote by FARGO-MP our solution with Multi-Probe as the probing strategy. We compare the performance of FARGO and FARGO-MP with the same number of candidate points T. In this setting, FARGO examines T points according to the global probing sequence; FARGO-MP examines T/L points in each hash table according to each local probing sequence. We plot the recall-time and ratio-time curves on *YahooMusic* when varying T – see Fig. 7. We observe that FARGO always takes less time than Multi-Probe to reach the same recall or overall ratio. This indicates that with the same number of candidate points, the results returned by FARGO are more accurate than those returned by FARGO-MP.

6.3 Comparison on Indexing Performance (Q4)

To evaluate the indexing performance of all algorithms, we compare the index size and indexing time in all datasets with default settings. The results are shown in Fig. 8. CeoMIPS fails to build indexes on *Trevi* since *Trevi* dataset has many identical points. (1) The index sizes of all algorithms except for ProMIPS depend directly on the number of indexes. FARGO only adopts $L = 5$ multi-dimensional hash tables and does not need to store the hash values, and thus always has the smallest index size. ProMIPS adopts an m-dimensional indexes ($m = 10$) but requires storing m projected values of all points, which brings it a larger index size than FARGO. H2-ALSH adopts $m = 62$ hash indexes for all datasets and always has the largest index size. (2) As for the indexing time, since FARGO, Simple-LSH and Range-LSH adopt hash tables as indexes, their construction time for each index is the lowest. Their total indexing time depends on the number of indexes. FARGO always achieves the lowest indexing time among all algorithms. H2-ALSH has a small indexing time because it just requires building indexes dimension by dimension. The indexing time of ProMIPS is also not high due to the small value of m. RPT and CeoMIPS have the much higher indexing time than other algorithms. The reason is that RPT needs to build the multi-dimensional partition-based trees, which is more time-consuming than other data structures. CeoMIPS needs to compute the extreme order statistics for $D = 1024$ projection functions, which incurs a extremely high computational cost.

6.4 Comparison on Query Performance (Q5, Q6)

6.4.1 ***Performance Overview.*** We report the query time (ms), overall ratio, and recall of all algorithms with default settings on all datasets in Table 3. FARGO has better query time than all competitors on all datasets, and the overall ratio and recall are also better than those of the competitors. We observe that *Cifar*, *Trevi* and *GIST* incur the smallest query time, which are less than 0.5 ms and several times smaller than our competitors. These datasets have different cardinalities, which indicates that the query time is not heavily affected by the dataset cardinality. This can be explained by the norm-ranging that eliminates the effect of cardinality. Another interesting observation is that the algorithms without norm-ranging, i.e., Simple-LSH, RPT, ProMIPS and CeoMIPS, have much higher query time than the algorithms with norm-ranging, i.e., FARGO, H2-ALSH, and Range-LSH, especially on large-scale dataset, such as *GIST*, *Tiny5M* and *Tiny80M*. This reveals that the norm-ranging plays an important role in MIPS. CeoMIPS fails to build indexes on *Trevi* since *Trevi* has many identical points.

6.4.2 ***Recall-Time and OverallRatio-Time Curves.*** We plot the recall-time and overall ratio-time curves for all algorithms on four datasets when varying c – see Figs. 9 – 10. The previous experiments show that FARGO, H2-ALSH, and Range-LSH are always the three best algorithms and perform much better than the others, so, we only report the results of these three algorithms. Since the results of these three algorithms have been almost close to the exact MIP, we adopt the log-log plot to show the Recall-Time and OverallRatio-Time curves to make the performance difference more clear. From the figures we can see that all algorithms return more accurate results with larger query times. FARGO again performs the best on all metrics, and H2-ALSH has the second best performance. They both perform much better than Range-LSH. When compared among different datasets, we find the query time of FARGO remains nearly unchanged with the data cardinality increasing. For example, the cardinality of *GIST* is 20 times of that of *MNIST*, while *GIST* incurs less query time than *MNIST* to reach the same recall. The query time of H2-ALSH and Range-LSH also increases gradually. This is because the norm-ranging allows us to terminate the query processing after only checking a few partitions. It suffices to find good results in enough points with large norms. Moreover, with the data cardinality increasing, the performance difference between FARGO and its competitors greatly increases. On *Tiny5M*, to reach the same recall or overall ratio, the query time of FARGO is one or two orders of magnitude lower than H2-ALSH and Range-LSH, which demonstrates the superiority of FARGO on large-scale datasets. This can be explained as follows: First, FARGO has a more accurate distance estimator than H2-ALSH and Range-LSH, which allows it to find more accurate result with checking the same number of points. Second, FARGO has a better transformation method and termination condition, which enables it to avoid examining unnecessary candidates when retrieved results are good enough.

7 RELATED WORK

The MIPS problem is a classic problem in the database community. Early methods employ space-partitioning trees, such as the M-tree [24] and cone-tree [36]. However, these methods are computationally expensive in high-dimensional spaces due to the curse of dimensionality. Therefore, lots of approximate solutions to the MIPS problem have been proposed [5, 23, 35, 38].

LSH is a mainstream approximation method for solving the MIPS problem. Since the inner product is not a metric, asymmetric transformation based methods are applied to convert the MIPS problem into NNS. Shrivastava et al. propose the L2-Transformation [38] and Correlation-Transformation [39]. However, these two transformations bring distortion errors, and thus limit the query performance. XBOX-Transformation [5] eliminates the distortion errors and is widely adopted by many studies, such as Simple-LSH [33], Range-LSH [45], H2-ALSH [19], RPT [23] and CeoMIPS [35]. Based on the observation that a large norm $\|x\|$ is more likely to lead to a large inner product $q^\top x$, Range-LSH [45] and H2-ALSH [19] develop the norm-ranging strategy to partition the dataset according to their norms, which allows us to quickly probe the high-quality MIPS results in a small part of partitions and greatly improve the query performance. In addition, many studies aim to terminate the query processing earlier based on the intermediate query status and demonstrate their effectiveness [26, 26, 29, 30, 40]. Li et al. [26] adopt a lightweight learning method to predict the minimal amount of search for each query to reach a target recall. Gogolou et al. [15] provide guarantees with probabilistic bounds along different dimensions for each query. These two studies train models with a small training set but bring a high improvement on query performance. EI-LSH [29, 30] designs a training-free adaptive termination on top

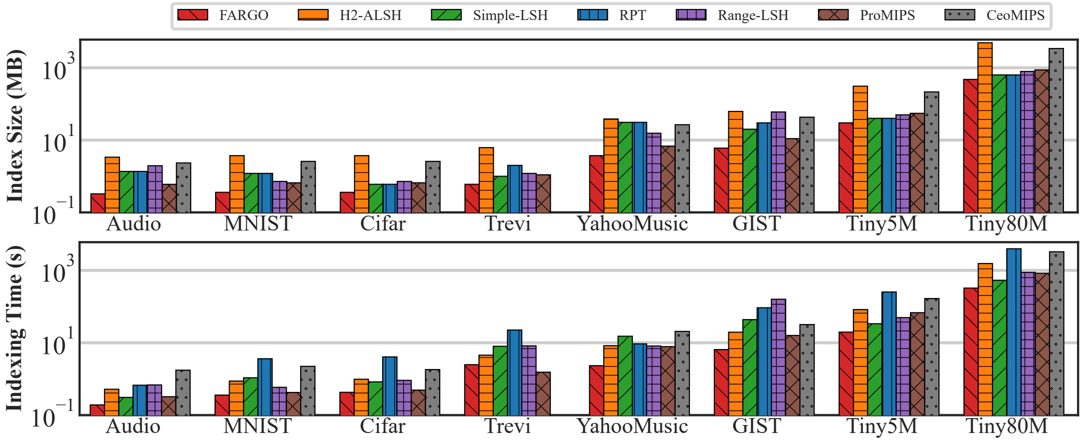
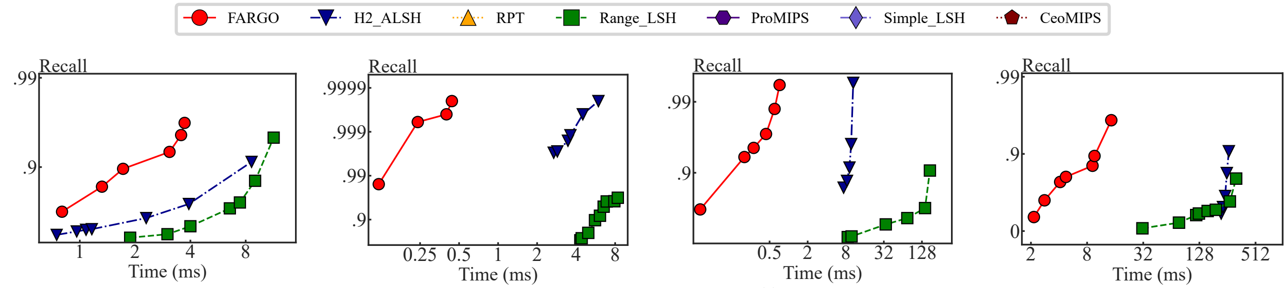
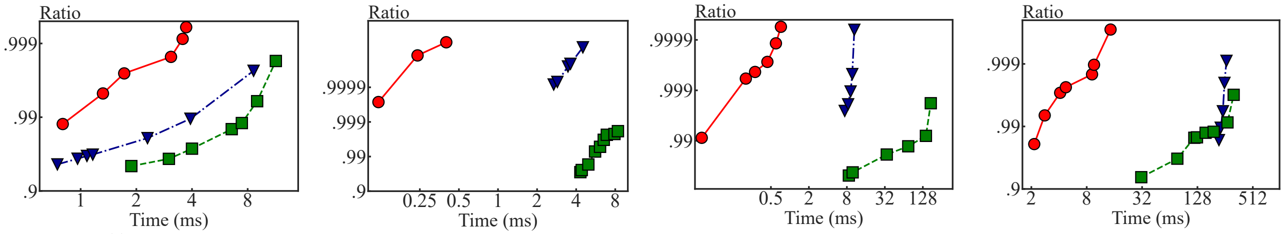


Figure 8: Indexing Performance on All Datasets



(a) MNIST (b) GIST (c) Tiny5M (d) Tiny80M

Figure 9: Recall-Time Curve



(a) MNIST (b) GIST (c) Tiny5M (d) Tiny80M

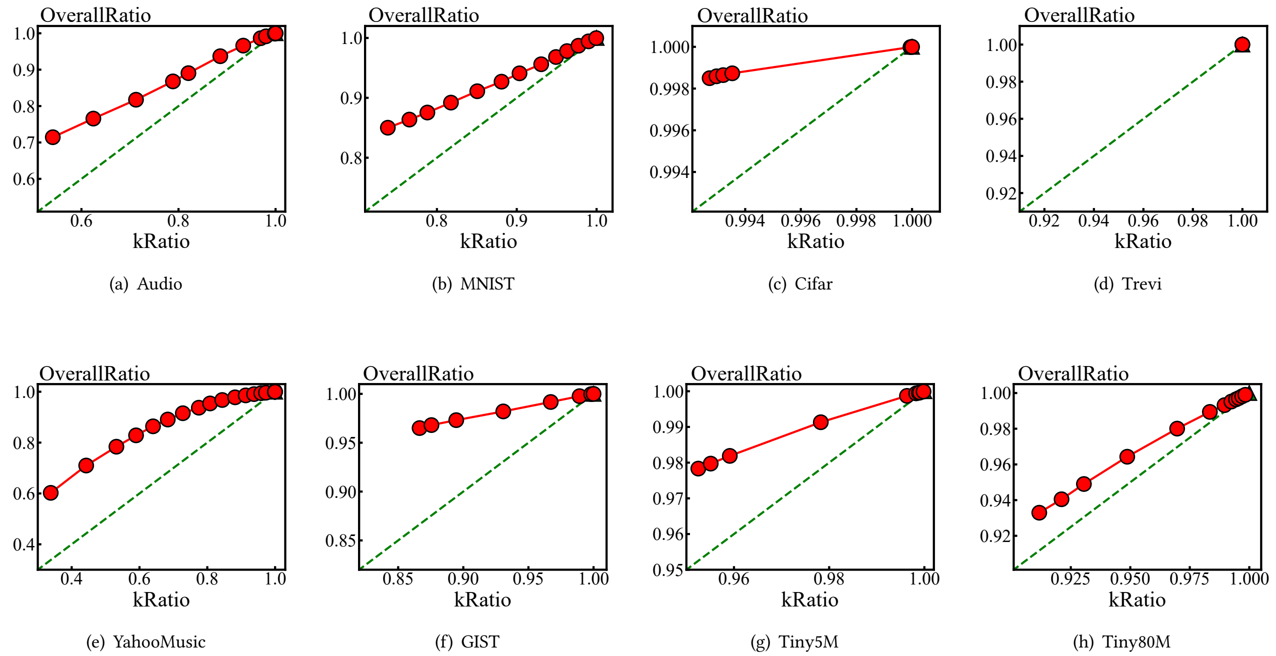
Figure 10: Ratio-Time Curve

of QA-LSH [18] by exploiting a smaller search radius in each dimension and exploring a tighter quality guarantee. The search radius in EI-LSH is set to constrain the hash value difference between q and x in each dimension. In contrast, FARGO probes the buckets based on the quantization distance, which is the weighted sum of the hash value differences between the query point q and the data point x. So the adaptive termination of EI-LSH is difficult to extend to FARGO and other LSH frameworks.

Recently, learning-based methods are proposed for MIPS [9, 13, 37]. Shen et al. [37] propose a binary code learning framework that preserves the inner product among raw data vectors. CeoMIPS [35] chooses a few projections associated with the extreme values of the query signature to answer MIPS problem. Moreover, neighbor graph-based methods are also proposed for MIPS [28, 32, 41] that were originally used to solve approximate NNS problems. However, both these learning-based and graph-based methods have extremely high training or construction cost, which is one or two order of magnitude higher than LSH methods and limits their usages on large-scale datasets.

8 CONCLUSION

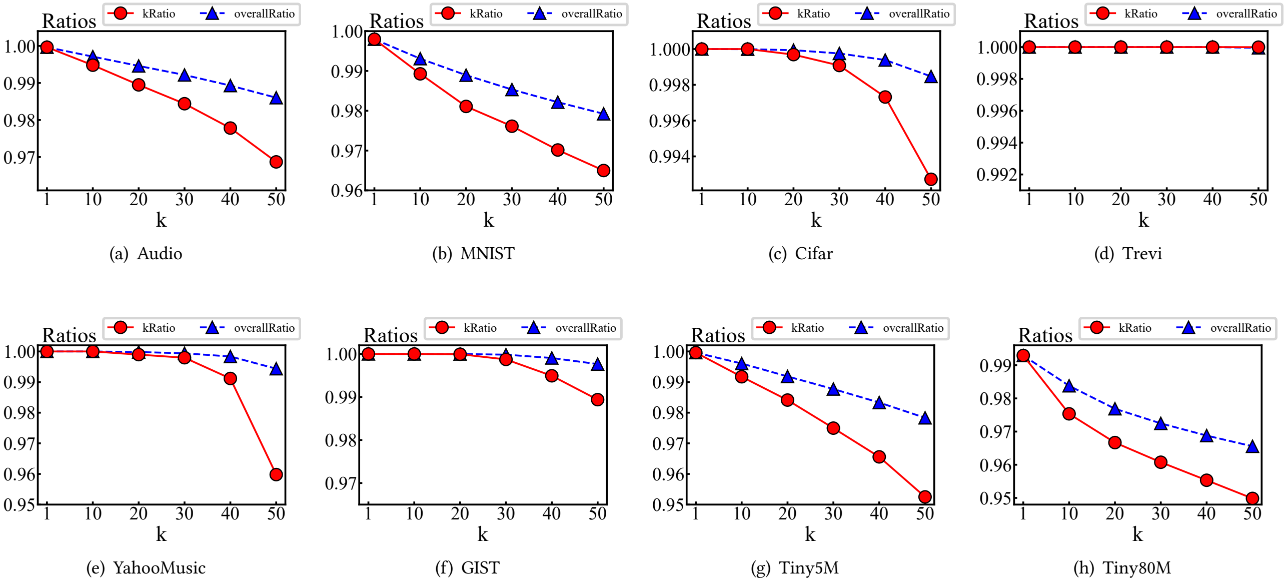
We propose FARGO for maximum inner product search with theoretical guarantees. First, we develop a novel transformation called RXT that converts this problem into approximate nearest neighbor search. Second, we propose a global multi-probing strategy as well as an early termination condition to examine high quality candidates. An experimental study shows that FARGO outperforms all its competitors in terms of both efficiency and accuracy. Specifically, FARGO improves the query time by an average of 35% when compared to the closest competitor. When all competitors take approximately the same query time, FARGO improves recall by 20–40% over the closest competitor.



(a) Audio (b) MNIST (c) Cifar (d) Trevi

(e) YahooMusic (f) GIST (g) Tiny5M (h) Tiny80M

Figure 11: kRatio-OverallRatio Curves



(a) Audio (b) MNIST (c) Cifar (d) Trevi

(e) YahooMusic (f) GIST (g) Tiny5M (h) Tiny80M

Figure 12: k-Ratios Curves

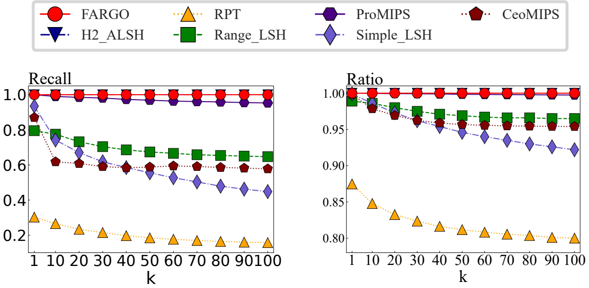
APPENDIX

To make the experimental results more convincing, we report the results of all sets of experiments in each dataset. Part A provides a comprehensive comparison between two metrics *OverallRatio* and *kRatio*. Part B shows the results of all datasets when comparing AET and NT, served as an extension of Section 6.2.2. Part C shows the results of all datasets when comparing GMP and Multi-Probe, served as an extension of Section 6.2.3. Part D discusses the experiment *Effect of* k. Part E shows Recall-Time and OverallRatio-Time curves on all datasets, served as an extension of Section 6.4.2.

A: Comparison between *OverallRatio* and *kRatio*

In this set of experiments, we compare two metrics *OverallRatio* and *kRatio*, where *kRatio* is defined as follows,

$$kRatio = \frac{q^\top x_k}{q^\top x_k^*}. \tag{20}$$



(a) Recall on GIST (b) OverRatio on GIST

Figure 13: Query Performance when Varying k

(1) We vary the approximate ratio c of FARGO to obtain different query performances and report the comparison between *OverallRatio* and *kRatio* at $k = 50$ in all datasets. The results are shown in Fig. 11, where the red line is the *kRatio-OverallRatio* curve and the green dash line is the line $y = x$ that helps us observe the difference between *OverallRatio* and *kRatio*. From Fig. 11, we can see that *OverallRatio* is always higher than *kRatio*, and the differences between *OverallRatio* and *kRatio* remain stable. (2) We compare *OverallRatio* and *kRatio* by conducting experiments when $k = 50$, and report the results when $k = \{1, 10, 20, 30, 40, 50\}$ in Fig. 12. We can see that both *OverallRatio* and *kRatio* decrease with k gradually and the differences between them increase with k. Even so, their largest difference (at $k = 50$) is still small.

From these two sets of experiments, the differences between *OverallRatio* and *kRatio* is small and stable. The differences among these *kRatio*s are also small. Therefore, So we think their functionalities are similar to evaluate the performance of the algorithms.

B: Comparison between AET and NT (Sec. 6.2.2)

We plot the recall-time and ratio-time curves in all datasets for FARGO and FARGO-NT – see Figs. 14-15. We observe that FARGO takes much less time than FARGO-NT to reach the same recall or overall ratio. Especially when the recall or overall ratio is high, the time difference is large. A key reason is that AET adaptively decides when to terminate so that fewer candidates are verified. Normally, to achieve a certain accuracy, the number of candidate points required to examine differs for each query point. NT does not consider this issue.

C: Comparison between GMP and Multi-Probe (Sec. 6.2.3)

We plot the recall-time and ratio-time curves in all datasets – see Figs. 16-17. We observe that FARGO always takes less time than Multi-Probe to reach the same recall or overall ratio. Especially in some datasets, such as *Audio* and *MNIST*, the performance difference between FARGO and FARGO-MP is huge. This indicates that with a same number of candidate points, the results returned by FARGO are more accurate than those returned by FARGO-MP.

D: Effect of k

We study the query performance when varying the value of k in $\{1, 10, 20, \cdots, 100\}$. In random projection algorithms, it is simple to answer kANN queries based on the original ANN queries. That is, we choose the best k results from the candidate sets rather than only the best result. Hence, the query time is affected little by k and is similar as that at $k = 50$, as shown in Table 3. Moreover, we find that as k increases, all algorithms achieve smaller overall ratio and recall. This is because given nearly the same number of points examined, a higher k makes it more likely to miss some exact results. The tendencies on all of datasets becomes extremely similar and thus we only report the performance in *GIST* in Fig. 13. In *GIST*, FARGO achieves the best performance on both *Recall* and *OverallRatio*. H2-ALSH is always the second-best. These two algorithms nearly achieve the recall of 1.0 and overall ratio of 1.0. ProMIPS also achieves very high recall (> 0.96), but it costs much higher query time than FARGO and H2-ALSH do (see Table 3).

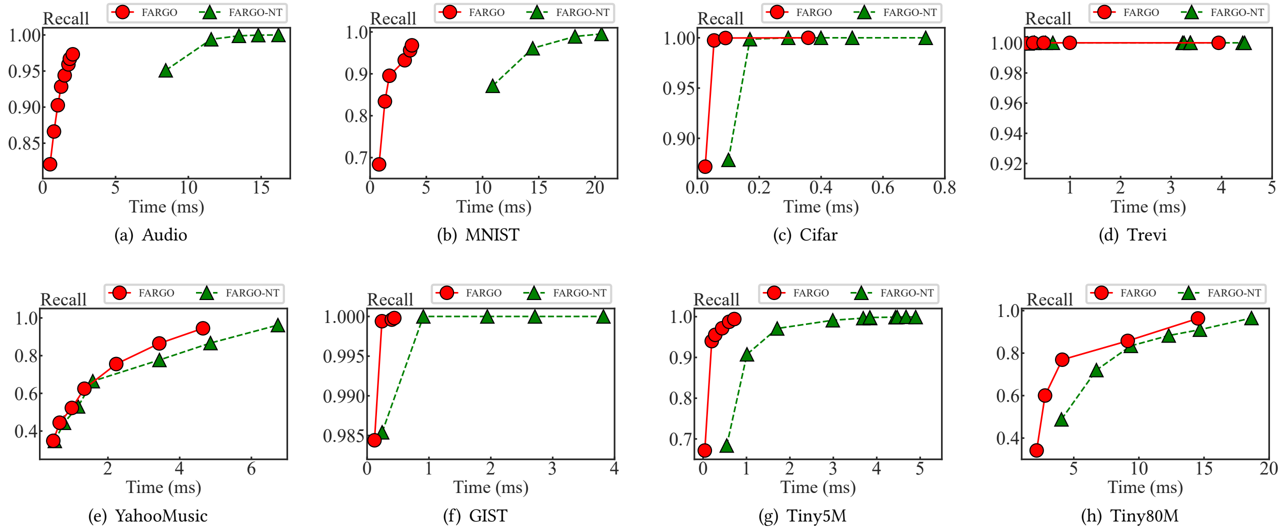
E: Recall-Time and OverallRatio-Time Curves (Sec. 6.4.2)

We plot the recall-time and overall ratio-time curves for all algorithms in all datasets when varying c – see Figs. 9 – 10. The previous experiments show that FARGO, H2-ALSH, and Range-LSH are always the three best algorithms and perform much better than the others, so we only report the results of these three algorithms. Since the results of these algorithms are almost close to the exact MIP, we adopt the log-log plot to show the Recall-Time and OverallRatio-Time curves to make the performance difference clearer. We can see that all algorithms return more accurate results with larger query times. FARGO again performs the best on all metrics, and H2-ALSH has the second best performance. They both perform much better than Range-LSH. When compared in different datasets, we find the query time of FARGO remains nearly unchanged with the increase of the data cardinality. For example, the cardinality of *GIST* is 20 times of that of *MNIST*, while *GIST* incurs less query time than *MNIST* to reach the same recall. On *Cifar*, *Trevi*, and *GIST*, FARGO and H2-ALSH quickly reach an extremely high query performance, i.e., the recall and ratio comes to 1, which implies it is rather easy to find the exact MIPS results in these datasets. The query time of H2-ALSH and Range-LSH also increases gradually. This is because the norm-ranging allows us to terminate the query processing after only checking a few partitions. It suffices to find good results in enough points with large norms. Moreover, with the data cardinality increasing, the performance difference between FARGO and its competitors greatly increases. In *Tiny5M*, to reach the same recall or overall ratio, the query time of FARGO is one or two orders of magnitude lower than H2-ALSH and Range-LSH, which demonstrates the superiority of FARGO on large-scale datasets. This can be explained as follows: First, FARGO has a more accurate distance estimator than H2-ALSH and Range-LSH, which allows it to find more accurate result with checking the same number of points. Second, FARGO has a better transformation method and termination condition, which enables it to avoid examining unnecessary candidates when retrieved results are good enough.

REFERENCES

[1] Mohamed Hussein Abdi, George Onyango Okeyo, and Ronald Waweru Mwangi. 2018. Matrix Factorization Techniques for Context-Aware Collaborative Filtering Recommender Systems: A Survey. *Sci* 11, 2 (2018), 1–10.

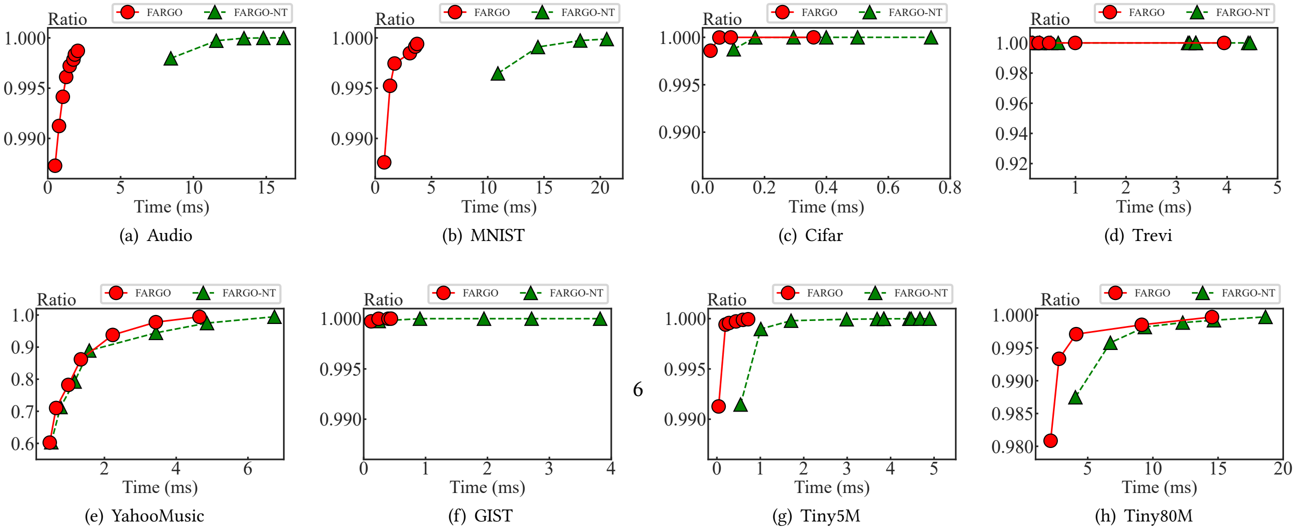
[2] Thomas Dybdahl Ahle, Rasmus Pagh, Ilya P. Razenshteyn, and Francesco Silvestri. 2016. On the Complexity of Inner Product Similarity Join. In *PODS*. 151–164.



(a) Audio (b) MNIST (c) Cifar (d) Trevi

(e) YahooMusic (f) GIST (g) Tiny5M (h) Tiny80M

Figure 14: Recall-Time Curves of FARGO and FARGO-NT



(a) Audio (b) MNIST (c) Cifar (d) Trevi

(e) YahooMusic (f) GIST (g) Tiny5M (h) Tiny80M

Figure 15: OverallRatio-Time Curves of FARGO and FARGO-NT

[3] Alexandr Andoni. 2004. E2lsh: Exact euclidean locality-sensitive hashing. *http://web. mit. edu/andoni/www/LSH/* (2004).

[4] Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya P. Razenshteyn. 2014. Beyond Locality-Sensitive Hashing. In *SODA*. 1018–1028.

[5] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. 2014. Speeding up the Xbox recommender system using a Euclidean transformation for inner-product spaces. In *RecSys*. 257–264.

[6] Robert M. Bell and Yehuda Koren. 2007. Lessons from the Netflix prize challenge. *SIGKDD Explorations* 9, 2 (2007), 75–79.

[7] Moses Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*. 380–388.

[8] Lijie Chen. 2018. On The Hardness of Approximate and Exact (Bichromatic) Maximum Inner Product. *Electronic Colloquium on Computational Complexity (ECCC)* 25 (2018), 26.

[9] Xinyan Dai, Xiao Yan, Kelvin Kai Wing Ng, Jiu Liu, and James Cheng. 2020. Norm-Explicit Quantization: Improving Vector Quantization for Maximum Inner Product Search. In *AAAI*. 51–58.

[10] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*. 253–262.

[11] Thomas L. Dean, Mark A. Ruzon, Mark Segal, Jonathon Shlens, Sudheendra Vijayanarasimhan, and Jay Yagnik. 2013. Fast, Accurate Detection of 100, 000 Object Classes on a Single Machine. In *CVPR*. 1814–1821.

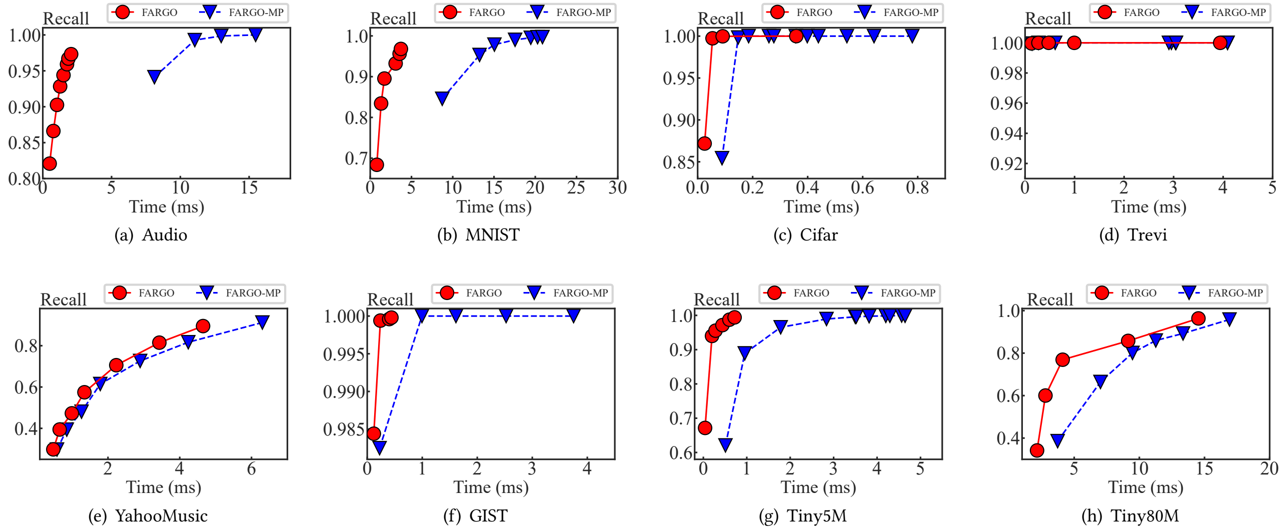
[12] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. 2012. The Yahoo! Music Dataset and KDD-Cup '11. In *KDD Cup (JMLR)*, Vol. 18. 8–18.

[13] Marco Fraccaro, Ulrich Paquet, and Ole Winther. 2016. Indexable Probabilistic Matrix Factorization for Maximum Inner Product Search. In *AAAI*. 1554–1560.

[14] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB*. 518–529.

[15] Anna Gogolou, Theophanis Tsandilas, Karima Echihabi, Anastasia Bezerianos, and Themis Palpanas. 2020. Data Series Progressive Similarity Search with Probabilistic Quality Guarantees. In *SIGMOD*. 1857–1873.

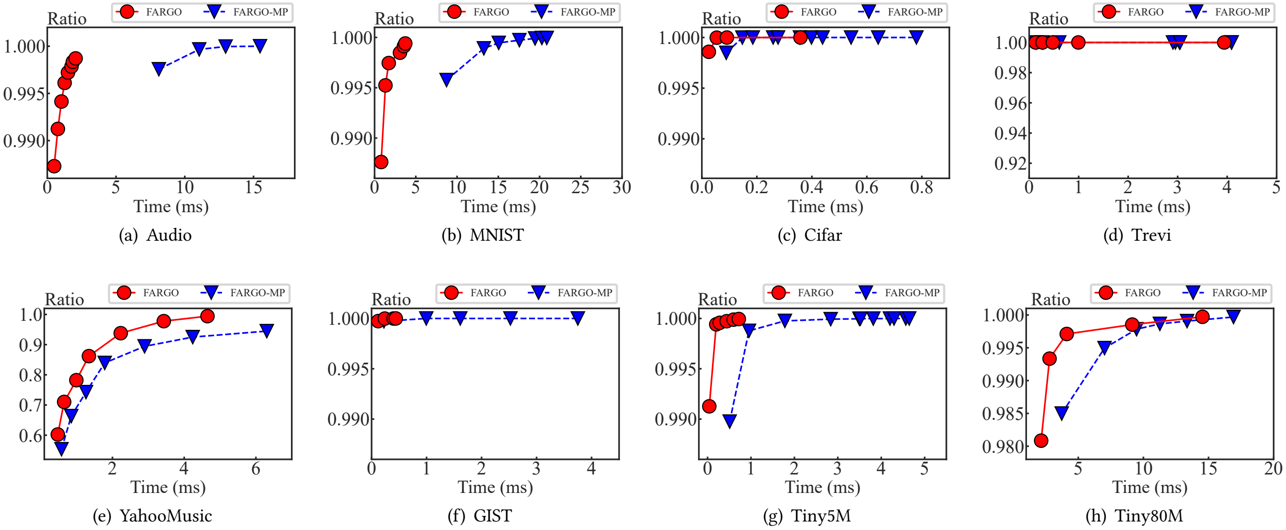
[16] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory Comput.* 8, 1 (2012), 321–350.



(a) Audio (b) MNIST (c) Cifar (d) Trevi

(e) YahooMusic (f) GIST (g) Tiny5M (h) Tiny80M

Figure 16: Recall-Time Curves of GMP and Multi-Probe



(a) Audio (b) MNIST (c) Cifar (d) Trevi

(e) YahooMusic (f) GIST (g) Tiny5M (h) Tiny80M

Figure 17: OverallRatio-Time Curves of GMP and Multi-Probe

[17] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.

[18] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search. *PVLDB* 9, 1 (2015), 1–12.

[19] Qiang Huang, Guihong Ma, Jianlin Feng, Qiong Fang, and Anthony K. H. Tung. 2018. Accurate and Fast Asymmetric Locality-Sensitive Hashing Scheme for Maximum Inner Product Search. In *KDD*. 1561–1570.

[20] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *STOC*. 604–613.

[21] Prateek Jain and Ashish Kapoor. 2009. Active learning for large multi-class problems. In *CVPR*. 762–769.

[22] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. 2009. Cutting-plane training of structural SVMs. *Mach. Learn.* 77, 1 (2009), 27–59.

[23] Omid Keivani, Kaushik Sinha, and Parikshit Ram. 2018. Improved maximum inner product search with better theoretical guarantee using randomized partition trees. *Machine Learning* 107, 6 (2018), 1069–1094.

[24] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*. 535–544.

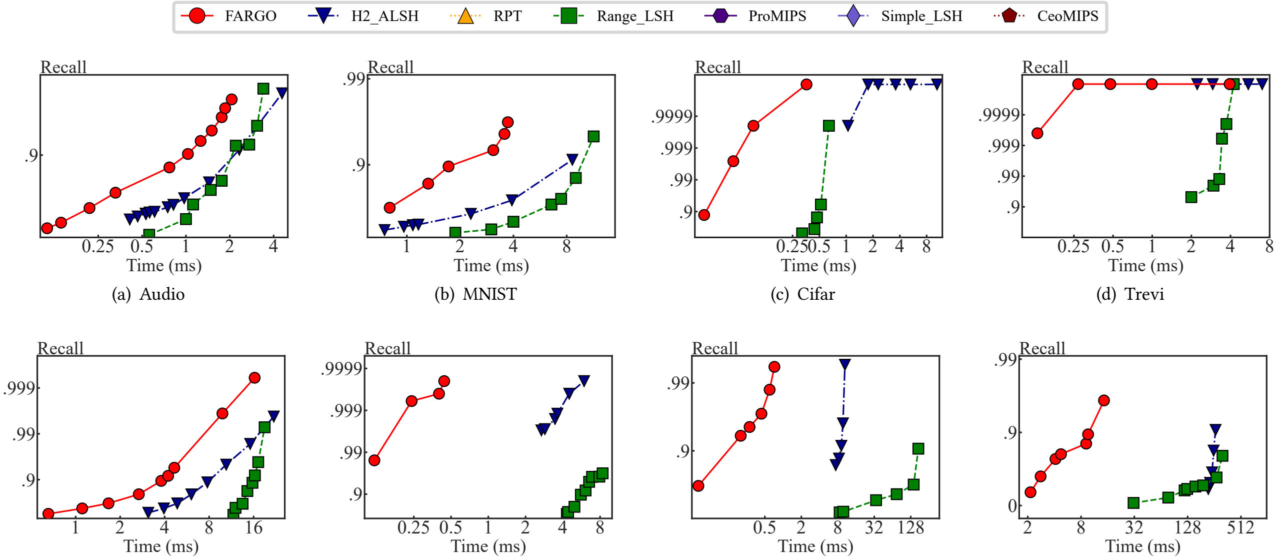
[25] Naama Kraus, David Carmel, Idit Keidar, and Meni Orenbach. 2016. NearBucket-LSH: Efficient Similarity Search in P2P Networks. In *SISAP (Lecture Notes in Computer Science)*, Vol. 9939. 236–249.

[26] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. 2020. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *SIGMOD*. 2539–2554.

[27] Jinfeng Li, Xiao Yan, Jian Zhang, An Xu, James Cheng, Jie Liu, Kelvin Kai Wing Ng, and Ti-chung Cheng. 2018. A General and Efficient Querying Method for Learning to Hash. In *SIGMOD*. 1333–1347.

[28] Jie Liu, Xiao Yan, Xinyan Dai, Zhirong Li, James Cheng, and Ming-Chang Yang. 2020. Understanding and Improving Proximity Graph Based Maximum Inner Product Search. In *AAAI*. 139–146.

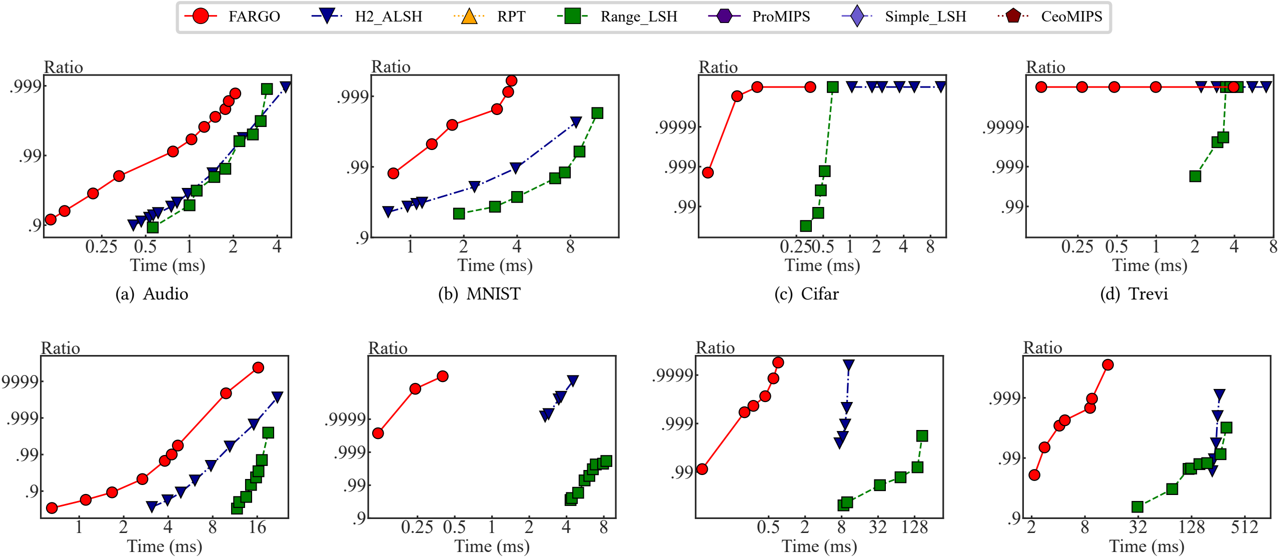
[29] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, and Lu Qin. 2019. I-LSH: I/O Efficient c-Approximate Nearest Neighbor Search in High-Dimensional Space. In *ICDE*. 1670–1673.



(a) Audio (b) MNIST (c) Cifar (d) Trevi

(e) YahooMusic (f) GIST (g) Tiny5M (h) Tiny80M

Figure 18: Recall-Time Curves



(a) Audio (b) MNIST (c) Cifar (d) Trevi

(e) YahooMusic (f) GIST (g) Tiny5M (h) Tiny80M

Figure 19: OverallRatio-Time Curves

[30] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, Lu Qin, and Xuemin Lin. 2021. EI-LSH: An early-termination driven I/O efficient incremental c-approximate nearest neighbor search. *VLDB J.* 30, 2 (2021), 215–235.

[31] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search. In *VLDB*. 950–961.

[32] Stanislav Morozov and Artem Babenko. 2018. Non-metric Similarity Graphs for Maximum Inner Product Search. In *NeurIPS*. 4726–4735.

[33] Behnam Neyshabur and Nathan Srebro. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *ICML*, Vol. 37. 1926–1934.

[34] Jennifer Nguyen and Mu Zhu. 2013. Content-boosted matrix factorization techniques for recommender systems. *Stat. Analysis and Data Mining* 6, 4 (2013), 286–301.

[35] Ninh Pham. 2021. Simple Yet Efficient Algorithms for Maximum Inner Product Search via Extreme Order Statistics. In *KDD*. ACM, 1339–1347.

[36] Parikshit Ram and Alexander G. Gray. 2012. Maximum inner-product search using cone trees. In *KDD*. 931–939.

[37] Fumin Shen, Wei Liu, Shaoting Zhang, Yang Yang, and Heng Tao Shen. 2015. Learning Binary Codes for Maximum Inner Product Search. In *ICCV*. 4148–4156.

[38] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS). In *NIPS*. 2321–2329.

[39] Anshumali Shrivastava and Ping Li. 2015. Improved Asymmetric Locality Sensitive Hashing (ALSH) for Maximum Inner Product Search (MIPS). In *UAI*. 812–821.

[40] Yang Song, Yu Gu, Rui Zhang, and Ge Yu. 2021. ProMIPS: Efficient High-Dimensional c-Approximate Maximum Inner Product Search with a Lightweight Index. In *ICDE*. IEEE, 1619–1630.

[41] Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2019. On Efficient Retrieval of Top Similarity Vectors. In *EMNLP/IJCNLP (1)*. 5235–5245.

[42] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2009. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*. 563–576.

[43] Christoph Haehling von Lanzenauer and William N Lundberg. 1974. The n-fold convolution of a mixed density and mass function. *ASTIN Bulletin: The Journal of the IAA* 8, 1 (1974), 91–103.

[44] Mengshuang Wang, Jun Ma, Shanshan Huang, and Peizhe Cheng. 2015. Combining Positive and Negative Feedbacks with Factored Similarity Matrix for Recommender Systems. In *WAIM*, Vol. 9098. 233–246.

[45] Xiao Yan, Jinfeng Li, Xinyan Dai, Hongzhi Chen, and James Cheng. 2018. Norm-Ranging LSH for Maximum Inner Product Search. In *NeurIPS*. 2956–2965.

[46] Bolong Zheng, Xi Zhao, 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. *PVLDB* 13, 5 (2020), 643–655.

[47] Yuxin Zheng, Qi Guo, Anthony K. H. Tung, and Sai Wu. 2016. LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index. In *SIGMOD*. 2023–2037.