Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search

JIUQI WEI, Institute of Computing Technology Chinese Academy of Sciences, University of Chinese Academy of Sciences, China
XIAODONG LEE, Institute of Computing Technology Chinese Academy of Sciences, Fuxi Institution, China
ZHENYU LIAO, Huazhong University of Science and Technology, China
THEMIS PALPANAS, LIPADE, Université Paris Cité, France
BOTAO PENG, Institute of Computing Technology Chinese Academy of Sciences, China

Approximate Nearest Neighbor (ANN) search in high-dimensional Euclidean spaces is a fundamental problem with a wide range of applications. However, there is currently no ANN method that performs well in both indexing and query answering performance, while providing rigorous theoretical guarantees for the quality of the answers. In this paper, we first design SC-score, a metric that we show follows the *Pareto principle* and can act as a proxy for the Euclidean distance between data points. Inspired by this, we propose a novel ANN search framework called *Subspace Collision (SC)*, which can provide theoretical guarantees on the quality of its results. We further propose SuCo, which achieves efficient and accurate ANN search by designing a clustering-based lightweight index and query strategies for our proposed subspace collision framework. Extensive experiments on real-world datasets demonstrate that both the indexing and query answering performance of SuCo outperform state-of-the-art ANN methods that can provide theoretical guarantees, performing 1-2 orders of magnitude faster query answering with only up to one-tenth of the index memory footprint. Moreover, SuCo achieves top performance (best for hard datasets) even when compared to methods that do not provide theoretical guarantees.

CCS Concepts: • **Information systems** → **Nearest-neighbor search**; **Query optimization**.

Additional Key Words and Phrases: Subspace collision, ANN search, High-Dimensional spaces

ACM Reference Format:
Jiuqi Wei, Xiaodong Lee, Zhenyu Liao, Themis Palpanas, and Botao Peng. 2025. Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 79 (February 2025), 29 pages. https://doi.org/10.1145/3709729

* Xiaodong Lee and Botao Peng are the corresponding authors.

Authors' Contact Information: Jiuqi Wei, Institute of Computing Technology Chinese Academy of Sciences, University of Chinese Academy of Sciences, China, weijiuqi19z@ict.ac.cn; Xiaodong Lee, Institute of Computing Technology Chinese Academy of Sciences, Fuxi Institution, China, xl@ict.ac.cn; Zhenyu Liao, Huazhong University of Science and Technology, China, zhenyu_liao@hust.edu.cn; Themis Palpanas, LIPADE, Université Paris Cité, France, themis@mi.parisdescartes.fr; Botao Peng, Institute of Computing Technology Chinese Academy of Sciences, China, pengbotao@ict.ac.cn.

ACM 2836-6573/2025/2-ART79
https://doi.org/10.1145/3709729

1 Introduction

Background and Problem. Nearest neighbor (NN) search in high-dimensional Euclidean spaces is a fundamental problem with various applications [30, 72, 75, 76] ranging from information retrieval [49], data mining [88], to recommender system [85]. Given a dataset $\mathcal{D}$ of n data points in d-dimensional space, a query $q \in \mathbb{R}^d$, and $k \in \mathbb{N}$, NN search will return the k nearest neighbors of q from $\mathcal{D}$. However, NN search in high-dimensional datasets is challenging due to the *curse of dimensionality* phenomenon [18, 25, 42, 56, 62, 101]. To achieve a better trade-off between query efficiency and accuracy, many researchers turn to Approximate Nearest Neighbor (ANN) search, sacrificing some query accuracy to achieve huge gains in efficiency [54, 90, 97]. ANN search will retrieve a set $\mathcal{C}$ of k high-quality candidates to maximize $recall = \frac{|\mathcal{C} \cap \mathcal{G}|}{k}$, where $\mathcal{G}$ consists of the exact k nearest neighbors of q in $\mathcal{D}$. Depending on where the index and dataset are stored, we can divide ANN methods into three categories: in-memory methods [14, 38, 67, 104], disk methods [53, 60], and memory-disk hybrid methods [24, 44]. In this paper, we focus on in-memory methods due to their great success for fast high-recall search.

Limitations and Motivation. Nowadays, new data is generated at an ever-increasing rate, and the size of datasets is also growing [55, 75, 76, 103]. We need to manage large-scale data more efficiently to support further data analysis [31, 37, 97, 98]. Many ANN search methods have been proposed, such as Locality-Sensitive Hashing (LSH)-based [4, 52, 91, 104, 111, 112], Vector Quantization (VQ)-based [9, 11, 38, 48, 74, 105], tree-based [14, 22, 50, 57, 70, 79, 80, 83, 93, 94, 99, 100, 106, 107, 113], and graph-based [6, 33, 34, 67, 71, 110]. However, each method has its own advantages and disadvantages. LSH-based methods are well-known for their robust theoretical guarantees on the quality of results, but they have to pay a high cost in terms of query answering time and have a large memory footprint [104]. VQ-based methods use the clustering structure as the index, so the index memory footprint is small, but a lot of indexing time is required to compute fine-grained clusters [9]. Tree-based methods can partition data points into different regions by splitting nodes, but as the space dimensionality increases, the effectiveness of trees decreases [17, 101], and query performance is thus limited. Graph-based methods typically have better query efficiency, but take considerably more time to construct indexes and require a larger memory footprint [31, 54]. The reason is that graph-based methods need to identify the near neighbors for each data point in the dataset (and connect to them) during the indexing phase, while in the query phase, they only need to search on a gradually converging path. Thus, no existing ANN method performs well in both indexing and query answering performance, while providing rigorous theoretical guarantees for the quality of the answers.

Our Method. In this paper, we propose a novel ANN search framework called *Subspace Collision (SC)* and design an index structure and query strategy for this framework, forming an efficient and accurate method named SuCo [102]. Compared to other ANN methods, SuCo performs well in both indexing performance (in terms of time and memory footprint) and query performance and has rigorous theoretical guarantees. Moreover, we first design SC-score, a metric that we show follows the *Pareto principle* (also known as the 80/20 rule) on many commonly used real-world datasets (Figure 2), demonstrating that data points closer to the query point tend to have larger SC-scores. As such, SC-score can act as a proxy for the Euclidean distance between data points. Second, inspired by SC-score, we present a novel ANN search framework called "Subspace Collision (SC)" (cf. Section 3) that is different from the existing LSH-based, VQ-based, tree-based, and graph-based frameworks. We design a naive method (without index structure) called SC-Linear, which achieves extremely high recall (over 0.99) on real-world datasets (cf. Table 2), illustrating the effectiveness of the subspace collision framework for ANN search. Third, we propose SuCo (cf. Section 4), which achieves efficient and accurate ANN search by designing an index structure and query strategies for

the subspace collision framework. SuCo constructs lightweight indexes by clustering data points in each subspace and using the inverted multi-index (IMI) to reduce the clustering complexity. We further design a new query algorithm for IMI, called *Dynamic Activation* (cf. Algorithm 3), which improves the query efficiency by up to 40% over the original query algorithm. Fourth, we perform a rigorous mathematical analysis, showing that the proposed subspace collision framework can provide theoretical guarantees on the quality of its results (cf. Theorems 1 and 2). Fifth, we conduct extensive experiments on real-world datasets. Numerical results show that SuCo outperforms state-of-the-art ANN methods that can provide theoretical guarantees in indexing and query answering performance, performing 1-2 orders of magnitude faster query answering with only up to one-tenth of the index memory footprint. Moreover, SuCo achieves top performance (best for hard datasets) even when compared to methods that do not provide theoretical guarantees.

Our main contributions are summarized as follows.

- We design SC-score, a metric that we show follows the *Pareto principle* on commonly used real-world datasets, demonstrating that SC-score can act as a proxy for the Euclidean distance between data points.
- We present *Subspace Collision (SC),* a novel ANN search framework based on SC-score. Our experiments show that the subspace collision framework can achieve high recall for ANN search. We perform rigorous mathematical analysis showing that the subspace collision framework can provide theoretical guarantees on the quality of its results.
- We propose SuCo (code available online [102]), which achieves efficient and accurate ANN search based on our subspace collision framework. We design a clustering-based lightweight index to ensure excellent indexing performance, and design query strategies to ensure excellent query performance.
- We conduct extensive experiments, demonstrating that both the indexing and query answering performance of SuCo outperforms state-of-the-art ANN methods that can provide theoretical guarantees, performing 1-2 orders of magnitude faster query answering with only up to one-tenth of the index memory footprint. Moreover, SuCo achieves top performance (best for hard datasets) even when compared to methods that do not provide theoretical guarantees.

2 Related Work

LSH-based methods. Locality-sensitive hashing (LSH)-based methods are known for their theoretical guarantees on the quality of returned query results [3, 4, 36, 43, 52, 59, 63, 65, 87, 91, 104, 111, 112]. A family of LSH functions is used to map data points from the original high-dimensional space to low-dimensional projected spaces. The properties of LSH can ensure that data points that are closer in the original space are also closer in the projected space [39]. For this reason, all that is needed to obtain high-quality results is to examine the points around the query point in the projected spaces [27]. LSH-based methods improve query efficiency by designing the index structure and query strategy in the projected spaces. Based on the query strategy, there are three categories of LSH-based methods: boundary constraint [3, 61, 89, 91], collision counting [36, 43, 52, 63, 65], and distance metric [87, 111]. DET-LSH [104] is the state-of-the-art LSH-based method, which combines the ideas of boundary constraint and distance metric.

VQ-based methods. Vector quantization is a lossy data compression technique that encodes vectors from a high-dimensional space into a finite set of values from low-dimensional discrete subspaces [40, 41]. Vector Quantization (VQ)-based methods aim to minimize the quantization distortion, which is the sum difference between each data point and its approximation [54]. Based on the way of combining quantizers among multiple subspaces, there are four categories of VQ-based

methods: product quantization (PQ) [45], additive quantization (AQ) [8], composite quantization (CQ) [109], and tree quantization (TQ) [10]. PQ is the most popular VQ-based methods, which divide the original high-dimensional space into a Cartesian product of many low-dimensional subspaces and then quantizes subvectors in each subspace separately [68]. Many extensions have been proposed to improve the performance of PQ for index construction and query answering [9, 11, 38, 46, 48, 69, 74, 105]. Although both PQ and subspace collision framework divide the high-dimensional space into subspaces, their design concepts are different. PQ sums the distances between quantized data points and the query in all subspaces to judge similarity, while the subspace collision framework counts the number of collisions between data points and the query in all subspaces to judge similarity. Optimized Product Quantization (OPQ) [38] using the inverted multi-index [9] is the state-of-the-art VQ-based method [31, 68].

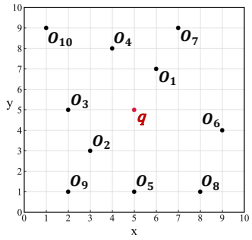
Tree-based methods. Tree-based methods hierarchically partition the high-dimensional space and group similar data points into the same partition as leaf nodes [97]. The partitioning process is usually recursive, so the root node of the tree covers the entire high-dimensional space, and its children nodes cover disjoint subspaces. Based on the way of partitioning, there are three categories of tree-based methods: pivoting [19, 21, 108], hyperplane [14, 20, 22, 26, 29, 32, 51, 58, 70, 81–83, 86, 93, 95, 96, 99, 107, 114] and compact partitioning [15, 35, 73]. Hyperplane partitioning is the most popular tree-based method, which recursively partitions the space by the hyperplane with random direction or axis-aligned separating hyperplane. Annoy [14] is the state-of-the-art tree-based method with balanced indexing and query answering performance [54].

Graph-based methods. Graph-based methods construct a proximity graph where each node represents a data point, and edges represent the neighbor relationships between data points [7, 90, 92]. The main idea of graph-based methods is "a neighbor's neighbor is likely also to be a neighbor". Compared with other types of methods, graph-based methods typically have better query efficiency but require considerably more time to construct indexes [31, 54]. The reason is that graph-based methods need to identify the near neighbors for each data point in the dataset (and connect to them) during the indexing phase, while in the query phase, they only need to search on a gradually converging path. Based on the way of building the proximity graph, there are three categories of graph-based methods: cluster and merge [71], iterate from an initial graph [23, 28, 33, 34, 54, 84], consecutive insertion [6, 64, 66, 67, 110]. Although there have been some innovative works subsequently, HNSW [67] is still one of the state-of-the-art graph-based method [92], which has an industrial-grade library and is therefore widely used.

3 Subspace Collision Framework

3.1 Framework Design

Intuitively, two data points that are close in the high-dimensional original space are also more likely to be close in its random subspace. Given a dataset $\mathcal{D}$ of n data points in d-dimensional space and a query $q \in \mathbb{R}^d$, and $o_1^*, \ldots, o_k^*$ are the k-NNs of q in $\mathcal{D}$. Suppose we randomly select s dimensions from all d dimensions as a subspace ($s < d$), and the points $o_1^*, \ldots, o_k^*$ and q in this subspace are denoted as $o_1^{*\prime}, \ldots, o_k^{*\prime}$ and q'. Then, it is expected that statistically, the proximity of $o_1^{*\prime}, \ldots, o_k^{*\prime}$ and q' is greater than other points in $\mathcal{D}$. The above intuition is based on the premise that the distances between data points are relatively evenly distributed in each dimension. If the distance between two points is concentrated in some dimensions, the above intuition may not hold. As shown in Figure 1(a) and Figure 1(b), if we choose the x dimension as the subspace for measuring distance, O_5 is the nearest neighbor because its distance from the query point q is only distributed in the y dimension. Similarly, if we choose the y dimension as the subspace for measuring distance, O_3 and O_6 are the nearest neighbors since their distances to q are mostly distributed in the x dimension.



(a) Data points distribution, where the Euclidean distance from O_1 to O_{10} to the query point q gradually increases.

x-axis as subspace

Rank	Point	Dist
1	o_5	0
2	o_1	1
3	o_4	1
4	o_2	2
5	o_7	2
⋮	⋮	⋮

y-axis as subspace

Rank	Point	Dist
1	o_3	0
2	o_6	1
3	o_1	2
4	o_2	2
5	o_4	3
⋮	⋮	⋮

(b) Nearest neighbors when choosing x or y dimension as the subspace for measuring similarity.

Rank	Point	SC-score
1	o_1	2
2	o_2	2
3	o_4	2
4	o_3	1
5	o_5	1
⋮	⋮	⋮

(c) Nearest neighbors when introducing the idea of collision based on x and y subspaces, where the collision ratio $\alpha = 0.5$.

Fig. 1. Illustration of finding nearest neighbors using the idea of subspace and collision.

Points like O_1 and O_2, which are close to q but their distances evenly distributed in both x and y dimensions, are easily missed.

We need to design a framework to solve the above problems, which meets two requirements:

- **Req1**: combine multiple subspaces to alleviate significant errors that a single subspace may cause;
- **Req2**: reduce the impact of distance ranking within a single subspace on the final result.

To this end, we first define "collision" and "subspace collision".

Definition 1 (Collision)**.** Given a dataset $\mathcal{D}$ of n data points in d-dimensional space, a query point $q \in \mathbb{R}^d$, and a collision ratio $\alpha \in (0, 1)$, if a data point $o \in \mathcal{D}$ satisfies: $\|o, q\|$ is one of the minimum $\alpha \cdot n$ distances between all n data points and q, i.e., o is one of the $(\alpha \cdot n)$-NNs of q in $\mathcal{D}$, we say that o collides with q.

Definition 2 (Subspace Collision)**.** Given a dataset $\mathcal{D}$ of n data points in d-dimensional space, a query $q \in \mathbb{R}^d$, and a collision ratio $\alpha \in (0, 1)$. We randomly select s dimensions from all d dimensions as a subspace $\mathbb{R}^s$ ($s < d$). The dataset $\mathcal{D}$, the data point o, and the query point q in this subspace are denoted as $\mathcal{D}'$, o', and q'. If a point $o \in \mathcal{D}$ satisfies: o' is one of the $(\alpha \cdot n)$-NNs of q' in $\mathcal{D}'$, we say that o collides with q in the subspace $\mathbb{R}^s$.

The definition of subspace collision downplays the distance ranking between data points and q in the subspace, allowing **Req2** to hold. As long as a data point is close enough to q (in the $(\alpha \cdot n)$-NNs), it can be considered as colliding with q in the subspace. Subspace collision can thus alleviate the problem that the distance between o and q is mainly distributed in specific dimensions.

Since using only one subspace to measure the similarity between data points and the query point may cause significant errors, we need to design a method to obtain multiple subspaces and a method that combines the collision results of these subspaces to judge the similarity. To make full use of the information in all d dimensions of the data points, we design a multi-round sampling strategy "Subspace Sampling" for the d dimensions.

Definition 3 (Subspace Sampling)**.** Given a dataset $\mathcal{D}$ of n data points in d-dimensional space, we adopt a multi-round sampling strategy to obtain N_s subspaces. In round i, a number of $s = \lfloor \frac{d}{N_s} \rfloor$ dimensions are uniformly sampled without replacement to form a subspace S_i, $i = 1, 2, \ldots, N_s - 1$. For the last subspace S_{N_s}, it simply pick up all remaining dimensions.

Table 1. Notations

Notation	Description
$\mathbb{R}^d$	d-dimensional Euclidean space
$\mathcal{D}$	Dataset of points in $\mathbb{R}^d$
n	Dataset cardinality $\lvert\mathcal{D}\rvert$
o, q	A data point in $\mathcal{D}$ and a query point in $\mathbb{R}^d$
o_i^*	The i-th nearest data point to q in $\mathcal{D}$
o_i	The i-th data point in $\mathcal{D}$
$\lVert o_1, o_2\rVert$	The Euclidean distance between o_1 and o_2
o', q'	o and q in a subspace
N_s	Number of subspaces
s	Dimension of each subspace
$S_i, \mathcal{D}_i$	The i-th subspace and all data points in S_i
o_j^i, q^i	The j-th data point and q in the i-th subspace
α	Collision ratio
β	Re-rank ratio
K, t	Number of K-means clusters and iterations

Definition 3 makes full use of all dimensions of the data. Even if $\frac{d}{N_s}$ is not an integer, all remaining dimensions after $N_s - 1$ rounds of sampling will be picked up by the last subspace S_{N_s}. For ease of reading and understanding, we assume that $\frac{d}{N_s}$ is an integer in the following sections, i.e., all subspaces $(S_1, ..., S_{N_s})$ have $s = \frac{d}{N_s}$.

Based on the obtained N_s subspaces, we design a metric "Subspace Collision Score (SC-score)" to measure the similarity between a data point and the query point.

Definition 4 (SC-score)**.** Given a dataset $\mathcal{D}$ of n data points in d-dimensional space, a query $q \in \mathbb{R}^d$, N_s s-dimensional subspaces, a collision ratio $\alpha \in (0, 1)$. Probe collisions of q in N_s subspaces with the collision ratio α. For a data point $o \in \mathcal{D}$, its SC-score is the number of subspaces where it collides with q. Therefore, SC-score is an integer in $[0, N_s]$.

SC-score combines the collision results of N_s subspaces to judge the similarity between data points and query points, allowing **Req1** to hold. As shown in Figure 1(c), data points that are closer to the query point in the original space tend to have larger SC-scores ($\alpha = 0.5$), and the subspace collision framework can return high-quality results even if the distances between data points and the query point are unevenly distributed. For example, O_1 and O_2 are not the closest points to q in both x and y dimensions, but they have the highest SC-score and, thus, the highest similarity. SC-score reduces the similarity measurement error caused by the unevenly distributed distance between o and q in different dimensions. The experimental results in Section 3.3.1 (Figure 2) show that SC-score is a good metric to measure the Euclidean distance between data points and query point because "*data points closer to query point tend to have larger SC-scores.*" Both Figure 1 and Figure 2 show that the proposed subspace collision framework appropriately addresses **Req1** and **Req2** simultaneously. Theorem 1 in Section 3.4 provides a theoretical guarantee on the effectiveness of SC-score.

3.2 Naive method for ANN

Based on the subspace collision framework proposed in Section 3.1, we design a naive method (without index structure) called SC-Linear to support ANN search, as shown in Algorithm 1. First,

Algorithm 1: SC-Linear

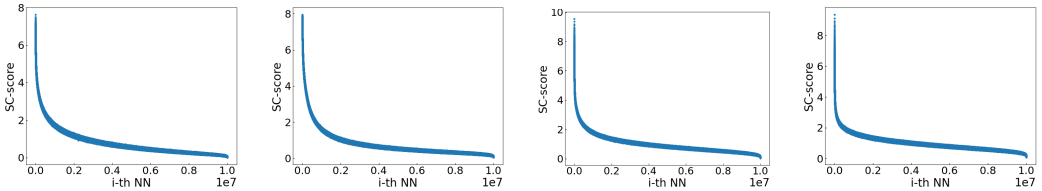
Input: Dataset $\mathcal{D}$, dataset size n, data dimensionality d, query point q, number of results k, subspace number N_s, collision ratio α, re-rank ratio β

Output: k nearest points to q in $\mathcal{D}$

1 Initialize an array *SC_scores* of length n and set all elements to 0;
2 Divide the d-dimensional space into N_s subspaces: $S_1, S_2, \ldots, S_{N_s}$;
3 Divide all data points $o_1, \ldots, o_n$ and q into N_s subspaces;
4 **for** $i = 1$ *to* N_s **do**
5 **for** $j = 1$ *to* n **do**
6 Calculate the Euclidean distance between o_j^i and q^i;
7 Sort $o_1^i, \ldots, o_n^i$ by their distances to q^i in S_i;
8 **for** $z = 1$ *to* $\alpha \cdot n$ **do**
9 Select the z-th-nearest point to q^i whose id in $\mathcal{D}$ is t;
10 *SC_scores*$[t]$++;
11 Sort *SC_scores* in descending order;
12 **for** $z = 1$ *to* $\beta \cdot n$ **do**
13 Select the point with the z-th-largest SC-score in *SC_scores* whose id in $\mathcal{D}$ is t;
14 Calculate the Euclidean distance between o_t and q;
15 **return** the *top-k* points closest to q in the $\beta \cdot n$ candidates;

an array is initialized to record the SC-score of each data point (line 1). Then, use Definition 3 to sample and obtain N_s subspaces, and divide all data points and the query point into these subspaces, where o_j^i and q^i represent the j-th data point and q in the i-th subspace (lines 2-3). In practice, for convenience, we can divide the d-dimensional space into N_s subspaces of the same length $s = \frac{d}{N_s}$, where the i-th subspace is allocated from the $(s \cdot (i-1)+1)$-th to $(s \cdot i)$-th dimensions in the original space, $i = 1, \ldots, N_s$. This division method can be regarded as a special case of Definition 3. In order to count the SC-score of each data point, it is necessary to calculate the Euclidean distance between each data point and q in each subspace (lines 4-6). For each subspace S_i, sort the distances and select $\alpha \cdot n$ points closest to q^i as collisions, and increase the SC-score of these points by one, where α is the collision ratio (lines 7-10). To further improve query accuracy, we design a re-ranking mechanism (lines 11-15). Specifically, sort the SC-scores of all points and select $\beta \cdot n$ points with the largest SC-scores as candidates, where β is the re-rank ratio (lines 11-13). Calculate the Euclidean distance between all candidates and q in the original space, and return the *top-k* points closest to q (lines 14-15). To facilitate understanding of SC-Linear, readers can refer to Figure 3 (the workflow of SuCo method). The difference between SC-Linear and SuCo is that SC-Linear does not construct the inverted multi-indexes to speed up query answering.

The experimental results in Section 3.3.2 show that SC-Linear has high query accuracy, which demonstrates the effectiveness of our designed subspace collision framework. Theorem 2 in Section 3.4 provides a theoretical guarantee on the quality guarantee of results returned by Algorithm 1. However, SC-Linear has no index structure to achieve fast query answering. It relies on calculating the Euclidean distance between all data points and the query point in all subspaces, which has the same time complexity as linear scanning in the original space (that is why we named Algorithm 1 SC-Linear). Therefore, we need to further design an index structure and query strategies for the subspace collision framework to improve query efficiency, which will be introduced in Section 4.



(a) Sift10M: d=128, N_s=8, s=16, α=0.1

(b) Yandex Deep10M: d=96, N_s=8, s=12, α=0.1

(c) Microsoft SPACEV10M: d=100, N_s=10, s=10, α=0.1

(d) Microsoft Turing-ANNS10M: d=100, N_s=10, s=10, α=0.1

Fig. 2. "Pareto principle" of SC-score on four datasets. Each figure contains n = 10M (10^7) scatter points, the scatter (i, j) represents the average SC-score of the i-th NN for 1000 queries is j, where $i = 1, 2, \ldots, 10^7$, and $j \in [0, N_s]$.

3.3 Preliminary Experiments

In this section, we conduct preliminary experiments to determine whether the subspace collision framework is effective. On the one hand, we explore whether SC-score is a good measure of the distance between all data points and the query point. On the other hand, we measure the performance of the SC-Linear algorithm to see whether it can return high-quality query results. More detailed experiments and parameter learning will be given in Section 5.3.

3.3.1 Effectiveness of SC-score. We select 1000 queries and count the SC-score of all data points in the dataset under each query. We perform this statistical analysis on multiple commonly used ANN datasets. Due to space limitations, we only show the results on four datasets in Figure 2 (other results are similar).

We can see from Figure 2 that SC-score follows the "Pareto principle" and is *insensitive* to the data distribution. Data points that are close to the query point have a high SC-score. Then, as the distance increases, the SC-score decreases rapidly until a "turning point" is reached. After the "turning point", the SC-score decreases slowly with the distance. The sharp change in trend makes each figure look like an "L" shape.

The "Pareto principle" of SC-score makes it very suitable for ANN search. In all four datasets, the x-axis of the "turning point" is about $0.2n$, which means that the 20% of data points closest to the query have a large and distinguishable SC-score, while the remaining 80% of the data points have a small and indistinguishable SC-score. Therefore, we only need to select data points with a large enough SC-score before the "turning point" to obtain high-quality ANN search results. SC-Linear is designed based on this idea. Specifically, the re-ranking mechanism (lines 11-15 in Algorithm 1) selects $\beta \cdot n$ data points with the largest SC-scores as candidates and then returns the *top-k* data points closest to the query among these candidates. Experimental results in Section 3.3.2 demonstrate that SC-Linear can return high-quality ANN results.

3.3.2 Performance of SC-Linear. To measure whether SC-Linear can return high-quality results, we perform ANN search when k=50 on four datasets (N_s and s set for each dataset are the same as that in Figure 2). Table 2 shows the average recall of 100 queries. We can see that SC-Linear can support accurate ANN search with extremely high recall, demonstrating that the subspace collision framework we designed is very suitable for ANN search. Furthermore, when the parameters N_s and α are set reasonably, SC-Linear can still have a high recall even if a very small re-rank ratio β is used. This also confirms the rule that "*data points closer to query point tend to have larger SC-scores*". More detailed parameter learning on α, β, and N_s will be given in Section 5.3.

Table 2. Recall of SC-Linear when returning 50 NNs under different datasets and parameters.

	α=0.05 β=0.001	α=0.05 β=0.005	α=0.05 β=0.01	α=0.05 β=0.05
Sift10M	0.9536	0.9916	0.9968	1
Deep10M	0.9418	0.9848	0.9968	1
SPACEV10M	0.9742	0.994	0.9962	0.9992
Turing10M	0.9638	0.9876	0.9944	0.999

3.4 Theoretical Guarantee

In this section, through rigorous mathematical analysis, we provide theoretical guarantees on the effectiveness of SC-score and the quality guarantee of our proposed subspace collision framework for ANN search.

Theorem 1 (Effectiveness of SC-score)**.** *Given a query point $q \in \mathbb{R}^d$, two independent random data points $o_1, o_2 \in$ dataset $\mathcal{D}$, and the SC-score of o_1 is greater than that of o_2, then $\|o_1, q\| < \|o_2, q\|$ holds with probability at least $1/2 - 1/e^2$ for appropriate choices of the number of subspaces N_s and collision ratio $\alpha \in (0, 1)$ that depend on the data statistics and the dimension d.*

Proof of Theorem 1. In the following, we provide a proof of Theorem 1 by making explicit the impact of the number of subspaces N_s and the collision ratio $\alpha \in (0, 1)$ as a function of the data statistics. Assume without loss of generality that $\frac{d}{N_s}$ is an integer. Let $o_i^j \in \mathbb{R}^{\frac{d}{N_s}}$ and q^j denote the subvector of o_i and q in the j-th subspace, with $i \in \{1, 2\}$, $j \in \{1, \ldots, N_s\}$, and N_s is the number of subspaces. Let $z_i = |o_i - q| \in \mathbb{R}^d$ denote the vector of absolute difference between o_i and q, and $z_i^j \in \mathbb{R}^{\frac{d}{N_s}}$ its subvectors. Assume that the squared Euclidean norms of subvector $Z_i^j \equiv \left\|z_i^j\right\|^2$ are independent random variables with mean $m > 0$ and variance σ^2, we compare the following differences

$$\|z_1\|^2 = \textstyle\sum_{j=1}^{N_s} Z_1^j, \quad \|z_2\|^2 = \sum_{j=1}^{N_s} Z_2^j, \tag{1}$$

by evaluating the number of subspace collisions (i.e., SC-score). Note that the subvectors o_1^j versus o_2^j with same index j *must* belong to one the following three scenarios:

(i) "subspace collision" happens for both subvectors, this happens $C \in \{0, \ldots, N_s - 1\}$ times; or
(ii) "subspace collision" happens for o_1^j but not o_2^j, this happens $\Delta \in \{1, \ldots, N_s - C\}$ times; or
(iii) "subspace collision" happens for neither subvector, this happens $N_s - C - \Delta$ times.

For scenario (i), it follows from the *Paley–Zygmund anti-concentration inequality* that,

$$\Pr\left(Z_i^j \le \sqrt{(1-\alpha)(\sigma^2 + m^2)}\right) \le \alpha, \tag{2}$$

with a collision ratio $\alpha \in (0, 1)$. We thus have, on the index set $\mathcal{S}_C$ (of cardinality C) on which collision occurs, that

$$0 \le \textstyle\sum_{j \in \mathcal{S}_C} Z_i^j \le C\sqrt{(1-\alpha)(\sigma^2 + m^2)}. \tag{3}$$

For scenario (iii), on non-collision subspaces (that corresponds to the index set $\mathcal{S}_{\neg C}$ having cardinality $N_s - C - \Delta$), note that the random variable $Z_1^j - Z_2^j$ is of zero mean and variance $2\sigma^2$, it then follows from *Chebyshev's inequality* that, for any $t > 0$,

$$\Pr\left(|Z_1^j - Z_2^j| \ge t\right) \le 2\sigma^2/t^2, \tag{4}$$

and thus

$$0 \le \textstyle\sum_{j \in \mathcal{S}_{\neg C}} |Z_1^j - Z_2^j| \le (N_s - C - \Delta)t. \tag{5}$$

For scenario (ii), we have, for some $t' > 0$ and a given j,

$$0 \le Z_1^j \le \sqrt{(1-\alpha)(\sigma^2+m^2)}, \quad \Pr\left(|Z_2^j - m| \ge t'\right) \le (\sigma/t')^2.$$

Finally, adding things up leads to

$$\sum_{j\in\mathcal{S}_C\cup\mathcal{S}_{\neg C}} |Z_2^j - Z_1^j| \le C\sqrt{(1-\alpha)(\sigma^2+m^2)} + (N_s - C - \Delta)t, \tag{6}$$

holds with probability at least $1 - 2(N_s - C - \Delta)\sigma^2/t^2$, for any $t > 0$. Note that the *worst case* is the case of $\Delta = 1$, taking then $t = c_1(m - \sqrt{(1-\alpha)(\sigma^2+m^2)})$ and $t' = c_2 m + (1-c_2)\sqrt{(1-\alpha)(\sigma^2+m^2)})$ with some $c_1, c_2 > 0$ such that

$$(c_2 - c_1(N_s - C - 1))m > (C + c_2 - c_1(N_s - C - 1))\sqrt{(1-\alpha)(\sigma^2+m^2)}, \tag{7}$$

then we have $\|o_1, q\| < \|o_2, q\|$. It can be checked that this happens with probability at least $1 - \frac{2(N_s-1)}{c_1^2}\left(m/\sigma - \sqrt{(1-\alpha)(1+m^2/\sigma^2)}\right)^{-2} - \left(c_2 \cdot m/\sigma + \sqrt{(1-\alpha)(1+m^2/\sigma^2)}(1-c_2)\right)^{-2}$. For given m, σ^2, we take $c_1 = \sqrt{8(N_s-1)}/(m/\sigma - \sqrt{(1-\alpha)(1+m^2/\sigma^2)})$, $c_2 = (e - \sqrt{(1-\alpha)(1+m^2/\sigma^2)})/(m/\sigma - \sqrt{(1-\alpha)(1+m^2/\sigma^2)})$ and any $\alpha > \max(1/(1+m^2/\sigma^2), 1 - e^2/(1+m^2/\sigma^2))$, we have $c_1, c_2 > 0$ so that the success probability is at least $1/2 - 1/e^2$. This concludes the proof of Theorem 1. □

Theorem 2 (Quality Guarantee of ANN Search)**.** *Given a query point $q \in \mathbb{R}^d$ and a dataset $\mathcal{D}$ consist of n independent random vectors, then, Algorithm 1 can answer a k-ANN query for q with probability at least $1/2$, with appropriate choices of the number of subspaces N_s, the collision ratio $\alpha \in (0,1)$, and the re-rank ratio $\beta \in (0,1)$, as a function of the data statistics, the dataset size n, and the dimension d.*

Proof of Theorem 2. Without loss of generality, we assume d/N_s is an integer. Following the notations in the proof of Theorem 1, we denote $o_i^j \in \mathbb{R}^{\frac{d}{N_s}}$ and q^j the subvector of o_i and q in the j-th subspace, respectively, with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, N_s\}$. Let $z_i = |o_i - q| \in \mathbb{R}^d$ denote the vector of absolute difference between o_i and q, and $z_i^j \in \mathbb{R}^{\frac{d}{N_s}}$ its subvectors. Assume that $Z_i^j \equiv \left\|z_i^j\right\|^2$ are independent random variables across index i and j, with mean $m > 0$ and variance σ^2, we obtain by independence that

$$\mathbb{E}[\|z_i\|^2] = N_s m, \quad \mathrm{Var}[\|z_i\|^2] = N_s \sigma^2, \tag{8}$$

for which each index j *must* belong to one the following two cases:

(i) "subspace collision" happens for subspace o_i^j, this happens for $C \in \{0, \ldots, N_s\}$ times for the data vector o_i; or
(ii) "subspace collision" does not happen for o_i^j, this happens for $N_s - C$ times for the data vector o_i.

Similar to the proof of Theorem 1, in each case, we can bound the value of Z_i^j as follows.

(i) By *Paley–Zygmund inequality* that

$$\Pr\left(Z_i^j \le \sqrt{(1-\alpha)(\sigma^2+m^2)}\right) \le \alpha, \tag{9}$$

with a collision ratio $\alpha \in (0,1)$.

(ii) By *Chebyshev's inequality* that, for any $t > 0$,

$$\Pr\left(|Z_i^j - m| \ge t\right) \le \sigma^2/t^2. \tag{10}$$

We thus have that

$$(N_s - C)(m - t) \leq \|z_i\|^2 \leq C\sqrt{(1-\alpha)(\sigma^2 + m^2)} + (N_s - C)(m + t),$$

holds with probability at least $1 - (N_s - C)\sigma^2/t^2$. In the following, without loss of generality, we only discuss here the case where the re-rank ratio β is chosen so that $C = N_s$, for which we should have k and β both small, and $\|z_i\|^2 \leq N_s\sqrt{(1-\alpha)(\sigma^2 + m^2)}$. Other scenarios with $C < N_s$ can be similarly studied by increasing β accordingly. In this setting, we show in the following, that the obtained range of the squared Euclidean distance $\|z_i\|^2 \in (0, N_s\sqrt{(1-\alpha)(\sigma^2 + m^2)}]$ is indeed among the top-k smallest in a set of n samples, with a controlled probability. The moments and joint distribution of the top-k ordered (e.g., smallest) values in a set of n i.i.d. samples has been extensively studied in the literature of *order statistics* [1, 2]. For the simplicity of exposition, we discuss here the case where the squared distances are normally distributed $\|z_i\|^2 \sim \mathcal{N}(N_s m, N_s\sigma^2)$ (which, as we shall see below, admits closed-form approximation for moments and leads to an explicit control of the success probability as a function of all tuning parameters). To treat general (e.g., non-Gaussian) distributions, it suffices to replace the (approximations of) first and second moments in (11) and (12) using PDF and CDF of the specific distribution. In the normal case, the expectation of the k-th order statistic (i.e., the expected value of the k smallest value) in a set of n sample with n large is approximately given by

$$E_{k,n} = N_s m + \sqrt{N_s\sigma^2} \cdot \Phi^{-1}\left(\frac{k - \gamma}{n - 2\gamma + 1}\right), \quad \gamma = 0.375, \tag{11}$$

and variance approximately given by

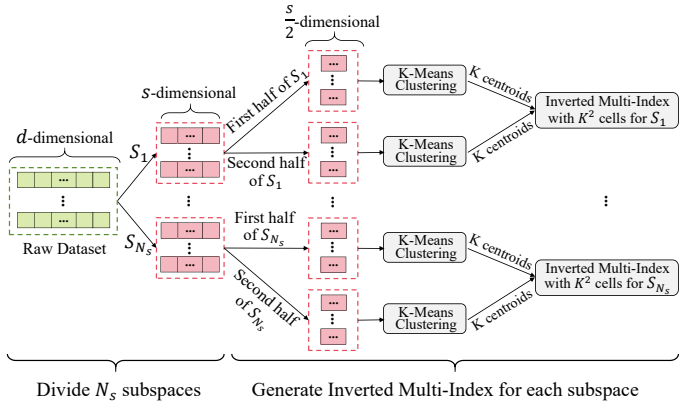
$$V_{k,n} = N_s\sigma^2 \cdot \frac{k(n-k+1)}{(n+1)^2(n+2)}\left(\phi\left(\Phi^{-1}\left(\frac{k}{n+1}\right)\right)\right)^{-2}, \tag{12}$$

with $\phi(\cdot)$ and $\Phi(\cdot)$ the PDF and CDF of standard Gaussian distribution. These approximations are known to be rather accurate as long as $n \geq 9$, see [13, 16]. Thus, it follows again from *Chebyshev's inequality* that the probability of correct answering is at least $1 - V_{k,n}/t^2$ for t such that $t > N_s m\sqrt{(1-\alpha)(1+\sigma^2/m^2)} - E_{k,n}$. Taking N_s and $\alpha \in (0,1)$ and $t = \sqrt{2N_s}\frac{\sigma}{n}\frac{k(n-k+1)}{n}\left(\phi\left(\Phi^{-1}\left(\frac{k}{n+1}\right)\right)\right)^{-1}$, we conclude that the probability of correct answering is at least $1/2$. This concludes the proof of Theorem 2. □

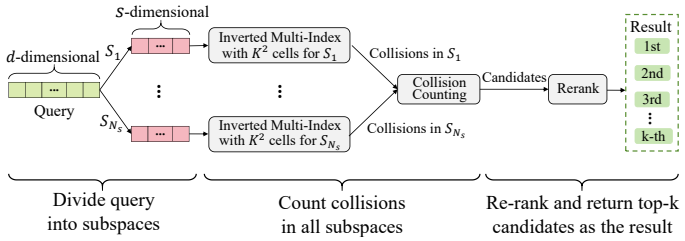
The proof of Theorem 1 relies on the idea that in cases where "subspace collision" happens for both or neither of the two subvectors, their Euclidean distance should be approximately the same; while in cases where "subspace collision" happens for one but not the other, their distance should differ by a significant amount, making the overall distance different. A similar argument holds for a single data in the proof of Theorem 2. Intuitively, larger values of collision ratio α and re-rank ratio β lead to higher computational complexity, while too small α and β reduce the accuracy of the proposed approach. As a consequence, the parameters α and β should be chosen (within a range) to achieve an optimal "computation-accuracy trade-off." This is consistent with the choice of α, β in the proofs of Theorems 1 and 2 above, and with our experimental conclusions in Section 5.3.3.

4 The SuCo Method

As analyzed in Section 3.2 and Section 3.3, SC-Linear's query efficiency is limited because when counting collisions in each subspace, it is necessary to calculate the Euclidean distance between the query and all data points (lines 4-10 in Algorithm 1). Therefore, the problem that needs to be solved is transformed into "*how to find the $\alpha \cdot n$ data points closest to the query in each subspace as quickly and accurately as possible.*"



(a) Index construction.



(b) Query answering.

Fig. 3. Overview of the SuCo workflow.

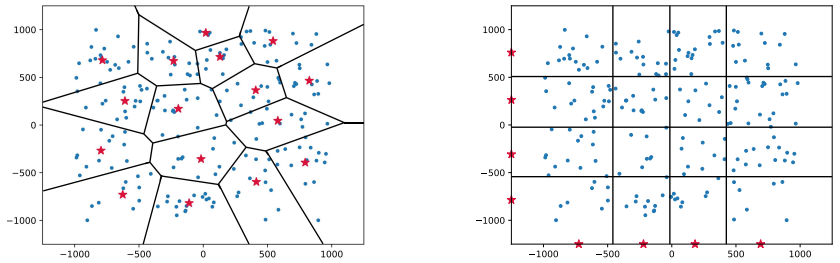
In this section, we propose SuCo to accelerate collision counting in each subspace by building lightweight indexes and designing query strategies. Figure 3 provides a high-level overview of the SuCo workflow, including index construction and query answering.

4.1 Index Construction

To count collisions as quickly and accurately as possible, i.e., find the $\alpha \cdot n$ data points closest to the query in each subspace, we need to construct an index in each subspace, which should be able to quickly locate data points around the query.

There are two requirements for the index structure: (1) The index structure should be lightweight enough. The index needs to be constructed independently in all N_s subspaces, and N_s is 6-12 in practice. If the index structure is complex, it requires a long indexing time and a large memory footprint to construct N_s indexes. Therefore, the graph-based and tree-based indexes that are currently popular in ANN search are no longer applicable. (2) The query method for collision counting needs to be simple and fast enough. Collision counting needs to be performed in all N_s subspaces, so the query speed of the index for each subspace needs to be fast enough. Fortunately, we don't need to obtain few (usually $k \in [1, 100]$) but accurate query results like graph-based and tree-based indexes. We only need to find many ($\alpha \cdot n$, usually $\alpha \in [0.01, 0.1]$) but not necessarily accurate data points as collisions.

Clustering-based indexes can meet the above two requirements. The index structure is lightweight enough because only the cluster centroids and data point assignments for each cluster need to



(a) K-means clustering with inverted index. K-means clustering is performed in the 2D space. The inverted index holds K 2D centroids (red stars in the figure).

(b) K-means clustering with inverted multi-index. K-means clustering is performed in each dimension. The inverted multi-index holds $\sqrt{K}$ 1D centroids for each dimension. K 2D clusters are obtained through Cartesian product of 1D clusters.

Fig. 4. Illustration of K-means clustering (K=16) in 2D space using inverted index or inverted multi-index.

be recorded. The query method for collision counting is simple and fast enough because only the distance between the query and each centroid needs to be calculated. Each centroid can represent all data points in its cluster, and if the centroid is close enough to the query, all data points within the cluster can be considered to collide with the query. Ordinary K-means clustering uses an inverted index to store which data points are in each cluster. However, to achieve fine-grained indexing, many clusters need to be constructed (K is large), which will restrict indexing and query answering efficiency. The inverted multi-index (IMI) [9] replaces the vector quantization inside the inverted index of K-means with the product quantization [45]. Figure 4 shows the difference between K-means clustering using the inverted index and the inverted multi-index. In Figure 4(a), K-means clustering is performed in the 2D space, and the inverted index holds K=16 2D centroids and the IDs of all data points belonging to each cluster. In Figure 4(b), K-means clustering is performed in each dimension, and the inverted multi-index holds $\sqrt{K} = 4$ 1D centroids for each dimension and the IDs of all data points belonging to the cluster. Then, K=16 2D clusters can be obtained through the Cartesian product of 1D clusters. Therefore, given the number of clusters K, K-means clustering using the inverted multi-index can reduce the time complexity from $O(K \cdot n \cdot d \cdot t)$ to $O(\sqrt{K} \cdot n \cdot d \cdot t)$ compared to using the inverted index, t is the number of iterations.

Figure 3(a) shows the workflow of SuCo to construct the index, and Algorithm 2 gives the pseudocode. First, we need to initialize two lists, one to store the centroids obtained by clustering in each subspace (the first part of our index) and the other to store the inverted multi-indexes constructed in each subspace (the second part of our index) (line 1). Then, we divide the d-dimensional original space into N_s subspaces $(S_1, S_2, \ldots, S_{N_s})$ with $s = \frac{d}{N_s}$ dimensions, and divide all data points into each subspace (lines 2-3). Assume without loss of generality that $\frac{s}{2}$ is an integer. For each subspace, based on the product quantization concept of IMI, we further divide each s-dimensional subspace into two $\frac{s}{2}$-dimensional subspaces (lines 4-7). Although each subspace can be divided into more than two parts, both the IMI paper [9] and our experiments show that dividing each space into two parts achieves the best performance. Then, for S_i, we perform K-means clustering on all data points in two $\frac{s}{2}$-dimensional subspaces (S_i^1 and S_i^2) to obtain the centroids of clusters $centroids_i^1, centroids_i^2$ and the data point assignments (belongs to which cluster) $assignments_i^1, assignments_i^2$ (lines 8-9). We put all the obtained centroids into a list as the first part of our index (line 10). To improve query efficiency, in each subspace S_i, we use the obtained $assignments_i^1, assignments_i^2$ to construct a map

Algorithm 2: Create Index

Input: A dataset $\mathcal{D}$, dataset size n, dimensionality of data points d, subspace number N_s, number of K-means clusters K, number of K-means iterations t

Output: Centroid list *centroids* and inverted multi-index list *IMIs*

1 Initialize a list *centroids* of length $2N_s$ and a list *IMIs* of length N_s;
2 Divide the d-dimensional space into N_s subspaces: $S_1, S_2, \ldots, S_{N_s}$;
3 Divide all data points into N_s subspaces: $\mathcal{D}_1, \ldots, \mathcal{D}_{N_s}$;
4 **for** $i = 1$ *to* N_s **do**
5 Initialize a map IMI_i as the inverted multi-index;
6 Further divide S_i into two subspaces S_i^1 and S_i^2;
7 Divide $\mathcal{D}_i$ into two subspaces: $\mathcal{D}_i^1$ and $\mathcal{D}_i^2$;
8 $centroids_i^1, assignments_i^1 \leftarrow$ **call** Kmeans$(\mathcal{D}_i^1, \sqrt{K}, t)$;
9 $centroids_i^2, assignments_i^2 \leftarrow$ **call** Kmeans$(\mathcal{D}_i^2, \sqrt{K}, t)$;
10 $centroids.append(centroids_i^1, centroids_i^2)$;
11 **for** $j = 0$ *to* $n - 1$ **do**
12 $IMI_i[assignments_i^1[j], assignments_i^2[j]].append(j)$;
13 $IMIs.append(IMI_i)$;
14 **return** *centroids* and *IMIs*;

IMI_i as the inverted multi-index, and we put all the obtained IMIs into a list as the second part of our index (lines 11-13). Finally, the centroids and IMI lists are returned as the SuCo indexes.

The original intention of IMI is to achieve as fine-grained data points division as possible, so K is very large ($K = 2^{28}$ is used in its paper [9]). In contrast, in the subspace collision framework, we only need to find a large number of data points (3%-5%) in the IMI of each subspace in a coarse-grained way, so K is much smaller ($K \in \left[2^{10}, 2^{12}\right]$). That is why SuCo's indexing performance outperforms all competitors (shortest time and least memory footprint); the experimental results are shown in Section 5.5.

4.2 Query Answering

To support ANN search based on the subspace collision framework, two query strategies need to design: (1) how to count collisions for the IMI of each subspace; (2) how to combine the collisions of IMIs in all subspaces to obtain the final ANN result.

The *Multi-sequence* algorithm was proposed with IMI to query IMI and obtain data points that are close to the query [9]. However, the *Multi-sequence* algorithm uses a priority queue to hold the candidate clusters, For the priority queue, both insertion and popping operations require logarithmic time complexity, which is time-consuming. Therefore, we design a new algorithm called *Dynamic Activation* to support query IMI without a priority queue. *Dynamic Activation* algorithm returns the same query results as *Multi-sequence* algorithm. The experimental results in Section 5.2 show that the efficiency of *Dynamic Activation* algorithm is up to 40% higher than that of the *Multi-sequence* algorithm.

Algorithm 3 gives the pseudocode of the *Dynamic Activation* algorithm. Given the distances between a query and all centroids in two subspaces $dists_1, dists_2$ and the indices of sorted distances in ascending order idx_1^i, idx_2^i, we need to select one cluster from each subspace to form a cluster in *IMI*. The sum of their distances (one from $dists_1$, the other from $dists_2$) is used as the selection criterion. The smaller the distance sum of clusters in *IMI*, the earlier they are retrieved. We first

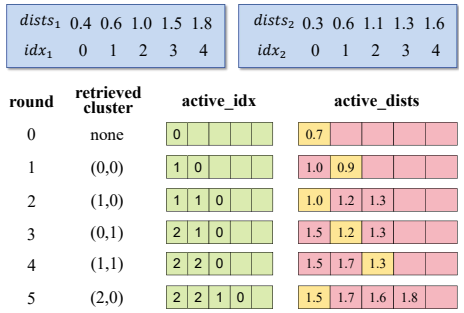


Fig. 5. An illustration of the *Dynamic Activation* algorithm

Algorithm 3: Dynamic Activation

Input: Collision ratio α, dataset size n, number of K-means clusters K, distances and indices of the first and second subspace $dists_1, idx_1, dists_2, idx_2$, the inverted multi-index IMI

Output: Clusters containing data points that collide with q

1 Initialize a list $retrieved\_clusters$, two arrays $active\_idx$ and $active\_dists$ of length $\sqrt{K}$;
2 $retrieved\_num \leftarrow 0$;
3 $active\_idx[0] \leftarrow 0$;
4 $active\_dists[0] \leftarrow dists_1[idx_1[0]] + dists_2[idx_2[0]]$;
5 **while** *TRUE* **do**
6 $pos \leftarrow$ index of the minimum element in $active\_dists$;
7 $cluster \leftarrow IMI[idx_1[pos], idx_2[active\_idx[pos]]]$;
8 $retrieved\_clusters.append(cluster)$;
9 $retrieved\_num$ += $sizeof(cluster)$;
10 **if** $retrieved\_num \geq \alpha \cdot n$ **then**
11 break;
12 **if** $active\_idx[pos] == 0$ *and* $pos < \sqrt{K} - 1$ **then**
13 $active\_idx[pos + 1] \leftarrow 0$;
14 $active\_dists[pos + 1] \leftarrow dists_1[idx_1[pos + 1]] + dists_2[idx_2[0]]$;
15 **if** $active\_idx[pos] < \sqrt{K} - 1$ **then**
16 $active\_idx[pos]$++;
17 $active\_dists[pos] \leftarrow dists_1[idx_1[pos]] + dists_2[idx_2[active\_idx[pos]]]$;
18 **return** $retrieved\_clusters$;

activate a cluster in the first subspace, which has the minimum distance in $dists_1$ (lines 3-4). Activated clusters can be combined with clusters from the second subspace, and the two clusters with the minimum sum of distances will be combined as a retrieved cluster in IMI (lines 6-9). When the number of data points in all retrieved clusters reaches the collision requirement (lines 10-11), the algorithm stops and returns the retrieved clusters (line 18). Otherwise, we dynamically activate the clusters in the first subspace in ascending order of $dists_1$ (lines 12-14) and update the combination information of the activated clusters (lines 15-17).

Figure 5 gives a running example of Algorithm 3. Algorithm 3 runs in a multi-round way, and in each round, it retrieves a cluster whose sum of $dists_1$ (among the activated indices) and $dists_2$ is

Algorithm 4: k-ANN Query

Input: A dataset $\mathcal{D}$, dataset size n, dimensionality of data points d, a query point q, number of results k, subspace number N_s, collision ratio α, re-rank ratio β, number of K-means clusters K, the centroid list $centroids$, the IMI list $IMIs$

Output: k nearest points to q in $\mathcal{D}$

1 Initialize an array SC_scores of length n and set to 0;
2 Divide q into N_s subspaces: $q^1, \ldots, q^{N_s}$;
3 **for** $i = 1$ *to* N_s **do**
4 Divide q^i into two subspaces q_1^i and q_2^i;
5 $centroids_1 \leftarrow centroids[2(i-1)]$;
6 $centroids_2 \leftarrow centroids[2(i-1)+1]$;
7 Calculate the distance between each centroid in $centroids_1(centroids_2)$ and $q_1^i(q_2^i)$, obtain $dists_1^i(dists_2^i)$;
8 Obtain the indices of sorted $dists_1^i(dists_2^i)$ in ascending order: $idx_1^i(idx_2^i)$;
9 $clusters \leftarrow$ **call** DynamicActivation $(\alpha, n, K, dists_1^i, idx_1^i, dists_2^i, idx_2^i, IMIs[i-1])$;
10 **for** *each cluster* $\in$ *clusters* **do**
11 **for** *each point_id* $\in$ *cluster* **do**
12 $SC\_scores[point\_id]$++;
13 Sort SC_scores in descending order;
14 **for** $z = 1$ *to* $\beta \cdot n$ **do**
15 Select the point with the z-th-largest SC-score in SC_scores whose index in $\mathcal{D}$ is t;
16 Calculate the Euclidean distance between o_t and q;
17 **return** the *top-k* points closest to q in the $\beta \cdot n$ candidates;

minimal. In the initialization round (round 0), the index 0 of $dists_1$ is activated (with initial value 0). In round 1, since only index 0 of $dists_1$ is activated, the cluster with the joint index of $dists_1$ and $dists_2$ (0, 0) is retrieved, and its distance to the query is 0.7. Then, we activate index 1 of $dists_1$ (with initial value 0) and increase the value of index 0 to 1. In round 2, two indices of $dists_1$ have been activated (0 and 1), and we retrieve the cluster with the joint index (1, 0) because it has a smaller distance (0.9) than the cluster with the joint index (0, 1). Then, we activate index 2 of $dists_1$ (with initial value 0) and increase the value of index 1 to 1. In round 3, three indices of $dists_1$ have been activated (0, 1, and 2), and we retrieve the cluster with the joint index (0, 1) because it has a minimum distance of 1.0. It should be noted that no new index of $dists_1$ is activated in this round, because for the retrieved cluster, the value of the activated index is not 0 but 1. The operations for subsequent rounds are the same as the previous rounds.

Algorithm 3 supports collision counting for the IMI of each subspace. We further design Algorithm 4 to combine the collisions of IMIs in all subspaces and support k-ANN queries, and Figure 3(b) shows the workflow of SuCo to answer k-ANN queries. First, we initialize an array to record the SC-score of all data points and divide the query q into N_s subspaces (lines 1-2). For each subspace, we calculate the distance between the query and all centroids in the two divided parts and then obtain the indices of sorted distances in ascending order (lines 3-8). Then, we call Algorithm 3 to obtain clusters containing data points that collide with q in each subspace and count collisions for all data points (lines 9-12). Finally, we re-rank the $\beta \cdot n$ data points with the largest SC-scores and return the *top-k* points closest to q among them (lines 13-17).

Table 3. Summary of datasets

Dataset	Cardinality	Dimensions	LID
Deep1M	1,000,000	256	37.26
Gist1M	1,000,000	960	70.15
Sift10M	10,000,000	128	22.05
Microsoft SPACEV10M	10,000,000	100	41.72
Yandex Deep10M	10,000,000	96	29.10
TinyImages80M	79.302,017	384	61.75
Sift100M	100,000,000	128	23.79
Yandex Deep100M	100,000,000	96	29.61

4.3 Complexity Analysis

For index construction, SuCo has time cost $O(n(\sqrt{K}dt + N_s))$ and space cost $O(\sqrt{K}d + nN_s)$. The time cost comes from three parts: (1) divide all data points into N_s subspaces, $O(nd)$; (2) use the K-means algorithm to cluster all data points within two parts of each subspace, $O(\sqrt{K}ndt)$; (3) build a map as the inverted multi-index in each subspace, $O(nN_s)$. Therefore, the total time cost is $O(n(\sqrt{K}dt + N_s))$. The space cost comes from two parts: (1) centroids obtained by clustering in each subspace, $O(\sqrt{K}d)$; (2) the inverted multi-indexes constructed in each subspace, $O(nN_s)$. Therefore, the total space cost is $O(\sqrt{K}d + nN_s)$.

For query answering, SuCo has time cost $O(n(\alpha N_s + \beta d + \log(\beta n)) + \sqrt{K}(d + \alpha\sqrt{K}\log\sqrt{K}))$. The time cost comes from seven parts: (1) calculate the distance between all centroids and the query point in each subspace, $O(\sqrt{K}d)$; (2) obtain the indices of sorted distances in ascending order in each subspace, $O(\sqrt{K}\log\sqrt{K})$; (3) invoke the *Dynamic Activation* algorithm to obtain clusters containing data points that collide with the query in each subspace, $O(\alpha K\log\sqrt{K})$; (4) count collisions for all data points in each subspace, $O(\alpha n N_s)$; (5) partial sort and select candidates with large SC-score, $O(n\log(\beta n))$; (6) calculate the distance between all candidates and the query, $O(\beta nd)$; (7) partial sort and return the *top-k* points closest to the query among candidates, $O(\beta n\log k)$. Therefore, the total time cost is $O(n(\alpha N_s + \beta d + \log(\beta n)) + \sqrt{K}(d + \alpha\sqrt{K}\log\sqrt{K}))$.

5 Experimental Evaluation

In this section, we first compare the performance of the *Dynamic Activation* algorithm we designed for IMI with the original *Multi-sequence* algorithm. Then, we study the performance of SuCo by self-evaluating and conducting comparative experiments with state-of-the-art ANN methods (with and without theoretical guarantees). All methods are implemented in C/C++. SuCo is implemented in C++ and compiled using -O3 optimization, useing OpenMP for parallelization and SIMD for accelerating calculations [102]. All experiments are conducted on a machine with 2 AMD EPYC 9554 CPUs @ 3.10GHz and 756 GB RAM, running on Ubuntu 22.04.

5.1 Experimental Setup

Datasets and Queries. We use eight real-world datasets for ANN search, whose key statistics are shown in Table 3. LID[1] is the local intrinsic dimensionality and a larger LID implies harder dataset. Note that the points in Sift10M and Sift100M are randomly chosen from the Sift1B dataset[2].

[1]There is no unified method to estimate the LID, and we use an open-source toolkit [12] with theoretical basis [47] to calculate it for each dataset.

[2]http://corpus-texmex.irisa.fr/

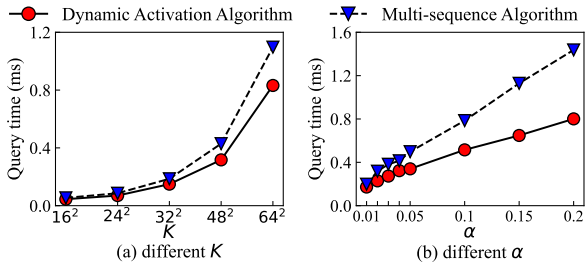


Fig. 6. Comparison of query efficiency between Dynamic Activation and Multi-sequence algorithm on Sift10M.

Similarly, the points in Microsoft SPACEV10M, Yandex Deep10M, and Yandex Deep100M are also randomly chosen from their 1B-scale datasets[3]. We randomly select 100 data points as queries and remove them from the original datasets.

Evaluation Measures. We adopt six measures to evaluate the performance of all methods: indexing time, index memory footprint, query time, query per second (QPS), recall, and mean relative error (MRE) [5, 77, 78], where the indexing time, query time, and QPS evaluate the efficiency of methods, the index memory footprint evaluates the storage resource consumption of methods, and the recall and MRE evaluate the quality of returned results. For a query q, if the returned result set is $R = \{o_1, \ldots, o_k\}$ and the exact k-NN set is $R^* = \{o_1^*, \ldots, o_k^*\}$, recall is defined as $\frac{|R \cap R^*|}{k}$, and mean relative error (MRE) is defined as $\frac{1}{k}\sum_{i=1}^{k} \frac{\|q,o_i\| - \|q,o_i^*\|}{\|q,o_i^*\|}$.

Benchmark Methods. We compare SuCo with seven state-of-the-art in-memory ANN methods. DET-LSH [104], DB-LSH [91], PM-LSH [111], and LCCS-LSH [52] are the state-of-the-art LSH-based methods that provide theoretical guarantees. They are single-threaded because the authors did not design parallel methods. OPQ [38], Annoy [14], HNSW [67], and SPTAG [23] are the state-of-the-art VQ-based, tree-based, graph-based, and tree-graph hybrid methods, respectively, which do not provide theoretical guarantees. They use OpenMP and/or Pthreads for parallelization, and SIMD for accelerating calculations.

Parameter Settings. Both the indexing and query answering phases of ANN methods require parameter settings. To select indexing parameters, we refer to the optimal parameters given in previous papers [31, 54, 104] and conduct our own experiments to verify their rationality so that all methods can achieve the best indexing performance. For DET-LSH, $L = 4$, $K = 16$. For DB-LSH, $L = 5$, $K = 12$. For PM-LSH, $s = 5$, $m = 15$. For LCCS-LSH, $m = 64$. For OPQ, $M = 2$, $K = 2^{20}$ (number of cells after Cartesian product). For Annoy, $f = 30$. For HNSW, $efConstruction = 200$, $M = 25$. For SuCo, $K = 50^2$, $N_s = 8$. For query parameters, we dynamically adjust their settings to observe the trade-off between query efficiency and accuracy (as shown in Figure 11 and Figure 12). For DET-LSH, DB-LSH, and PM-LSH, $\beta \in [0.005, 0.2]$, $c = 1.5$. For LCCS-LSH, $check\_k \in \left[2^8, 2^{18}\right]$. For OPQ, $\beta \in \left[10^{-4}, 10^{-1}\right]$. For Annoy, $search\_k \in \left[10^3, 10^5\right]$. For HNSW, $efSearch \in [300, 3000]$. For SPTAG, we use the balanced K-means tree (BKT). For SuCo, $\alpha \in [0.01, 0.1]$, $\beta \in [0.001, 0.05]$. k in k-ANN is set to 50.

5.2 Dynamic Activation vs. Multi-sequence

As introduced in Section 4.2, we design a new query strategy (Algorithm 3) to support efficient querying of IMI. Figure 6 compares the query efficiency between *Dynamic Activation* algorithm (we

[3]https://big-ann-benchmarks.com/neurips21.html

Table 4. Comparison on SuCo and SC-Linear

Method	Dataset	Query Time (ms)	Speedup	Recall
SC-Linear	Sift10M	3104.42	\	0.968
	Sift100M	71644.5	\	0.9942
SuCo	Sift10M	5.139	604.1	0.9346
	Sift100M	68.835	1040.8	0.9822

designed) and *Multi-sequence* algorithm (proposed with IMI [9]). While achieving the same query accuracy (they return the same query results), the efficiency of *Dynamic Activation* algorithm is up to 40% higher than that of the *Multi-sequence* algorithm. The advantage of *Dynamic Activation* algorithm increases as the query workload increases (larger K and α). The reason is that the *Multi-sequence* algorithm relies on a priority queue to hold the candidate clusters, and frequent insertion and popping operations are time-consuming. *Dynamic Activation* algorithm updates and maintains the activation list through activation strategies, eliminating the reliance on the priority queue and improving query efficiency.

5.3 Self-evaluation of SuCo

5.3.1 SuCo vs. SC-Linear. Preliminary experiments in Section 3.3 have demonstrated that the subspace collision framework is effective, and Table 2 shows that SC-Linear (has no index) can return high-quality query results. In this section, we further compare the query performance between SuCo and SC-Linear under the same parameter settings: $\alpha = 0.03$, $\beta = 0.003$. As shown in Table 4, compared with SC-Linear, the query efficiency of SuCo is significantly improved (up to 1000 times), with an acceptable sacrifice of query accuracy. The results demonstrate that the index structure and query strategy we designed for SuCo can be well applied to the subspace collision framework for ANN search.

5.3.2 Parameter study on K and N_s. Parameters K and N_s determine the efficiency and space consumption of index construction and affect the efficiency and accuracy of queries. We evaluate the impact of K and N_s on the performance of SuCo by setting K in the range $\left[2^8, 2^{16}\right]$ and N_s in the range $[6, 16]$. As shown in Figure 7 (a) and (b), the indexing time and query time increase gradually with K increasing from 2^8 to 2^{12} but significantly with K increasing from 2^{12} to 2^{16}. In contrast, the recall increase significantly with K increasing from 2^8 to 2^{12} but gradually with K increasing from 2^{12} to 2^{16}. The index memory footprint remains rather stable in the whole range. Therefore, we choose $K = 50^2 \in (2^{10}, 2^{12})$ as the default value. As shown in Figure 7 (c) and (d), the indexing time, index memory footprint, and query time continue to increase as N_s increases. The recall remains rather stable when N_s is larger than 8. Considering these two factors, we choose $N_s = 8$ as the default value. Based on our experiments with datasets of different dimensionality and size, we suggest choosing $N_s \in [6, 12]$ for new datasets, which is a good balance between indexing cost and query accuracy.

5.3.3 Parameter study on α and β. Parameters α and β have a great impact on query efficiency and accuracy. We evaluate the impact of α and β on the query performance of SuCo by setting α in the range $[0.01, 0.2]$ and β in the range $[0.001, 0.009]$. As shown in Figure 8 (a) and (b), the query time continues to increase as α increases, but the recall is rather stable when α is larger than 0.05. As shown in Figure 8 (c) and (d), the query time continues to increase as β increases, but the recall increases slowly when β is larger than 0.005. Therefore, $\alpha = 0.05, \beta = 0.005$ is a

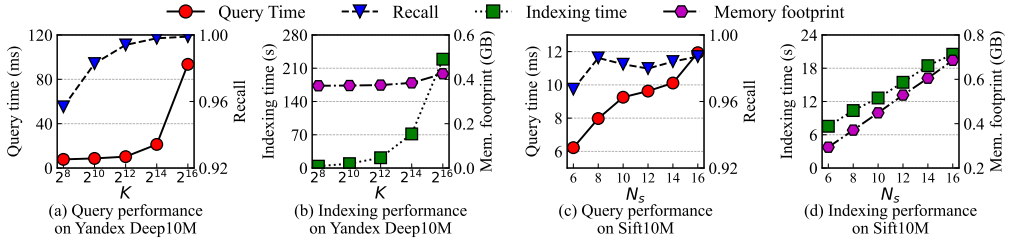


Fig. 7. Performance of SuCo when varying the number of K-means clusters K and the number of subspaces N_s.

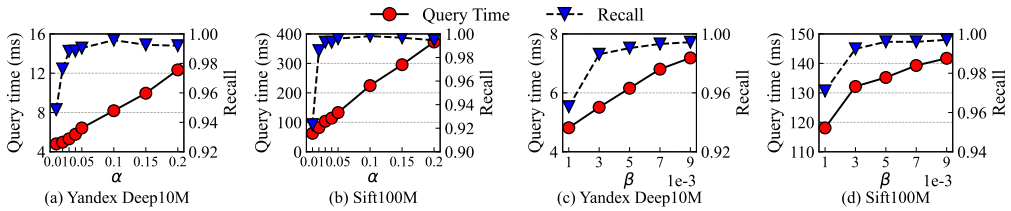


Fig. 8. Query performance of SuCo when varying the collision ratio α and the re-rank ratio β.

Table 5. SuCo under different distance measures

		Gist 1M	Sift 10M	SPACEV 10M	Yandex Deep 100M
SuCo-L1	Recall	0.9328	0.9868	0.966	0.9906
	MRE	0.00099	0.00039	0.00103	0.0002
SuCo-L2	Recall	0.9508	0.9862	0.9802	0.9988
	MRE	0.00068	0.00042	0.00035	0.00002

good choice for SuCo. In addition, we found that SuCo can achieve a high recall with smaller α and β on large-scale datasets. On the Sift100M dataset, the recall of SuCo is 0.9822 when $\alpha = 0.03$, $\beta = 0.003$. This demonstrates that SuCo has good scalability on large-scale datasets. Based on our experiments with datasets of different dimensionality and size, we suggest choosing $\alpha \in [0.03, 0.1]$ and $\beta \in [0.003, 0.005]$ for new datasets, which is a good balance between query efficiency and accuracy. In addition, large-scale datasets tend to use smaller α and β, and hard datasets tend to use larger β.

5.3.4 SuCo under different distance measures. In this paper, we focus on the L2 distance (Euclidean distance), which is the most widely used distance measure. It is interesting to examine if SuCo can be used with other distance measures. Table 5 reports the performance of SuCo using the L1 (Manhattan distance) and L2 distance measures on four datasets under the same setting. We observe that SuCo achieves high query accuracy with L1, too.

5.4 SuCo vs. Competitors with Guarantees

5.4.1 Indexing performance. Figure 9 shows the indexing time and index memory footprint of all methods that provide theoretical guarantees. We found that SuCo and single-threaded SuCo have the least index memory footprint. Since SuCo's clustering-based index structure is lightweight and

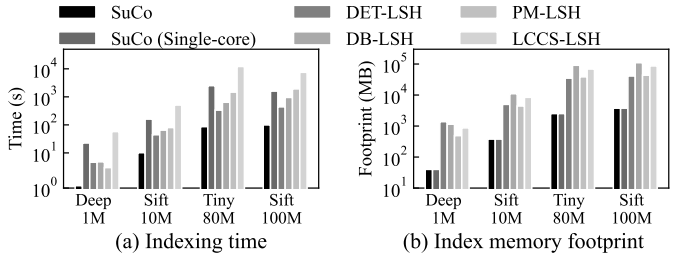


Fig. 9. Indexing performance comparison between SuCo and competitors that provide theoretical guarantees.

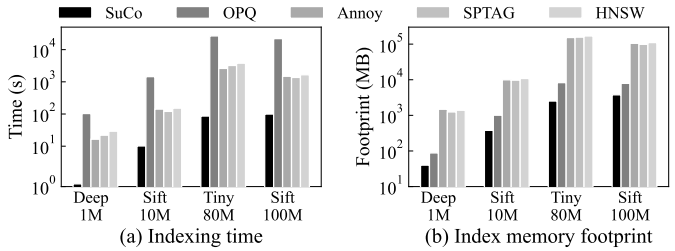


Fig. 10. Indexing performance comparison between SuCo and competitors that do not provide theoretical guarantees.

only the cluster centroids and data point assignments need to be recorded, the required memory space is small. In addition, SuCo has the best indexing efficiency, while single-threaded SuCo is (naturally) slower, with the extra cost coming from the (now serial) subspace clustering step. In practice, parallel indexing methods are used in the modern multi-core systems to speed up index construction, so SuCo has an advantage in indexing efficiency compared to other methods with theoretical guarantees. LSH-based methods need to build one (PM-LSH) or multiple (DET-LSH, DB-LSH) index trees or circular shift array (LCCS-LSH), which are heavy-weight index structures, requiring more time to construct the index, and occupying more memory.

5.4.2 Query performance. For all methods, there is a trade-off between query efficiency and accuracy. Figure 11 gives the recall-QPS and MRE-QPS curves of all methods that have theoretical guarantees. We found that the query performance of SuCo and single-threaded SuCo is 1-2 orders of magnitude better than that of LSH-based methods. Benefiting from the design of the subspace collision framework, index structure, and query strategy, SuCo can efficiently count collisions to obtain candidate points and then ensure the high quality of query results through a re-ranking mechanism. SuCo only needs to retrieve 0.3%-0.5% of candidate points to achieve a high recall, indicating that the obtained candidate points are of high quality. However, DET-LSH, DB-LSH, and PM-LSH need to obtain more candidate points (up to 10%) and spend more time to achieve the same recall because the LSH projection loses the distance information between data points.

5.5 SuCo vs. Competitors without Guarantees

5.5.1 Indexing performance. Figure 10 shows the indexing time and index memory footprint of SuCo and competitors that do not provide theoretical guarantees. We found that SuCo has the best indexing efficiency and the least index memory footprint. Since SuCo only needs to find many (about 3%-5%) but not necessarily accurate data points as collisions, the index structure does not need to divide the data points very finely. To construct the IMI of each subspace, we only

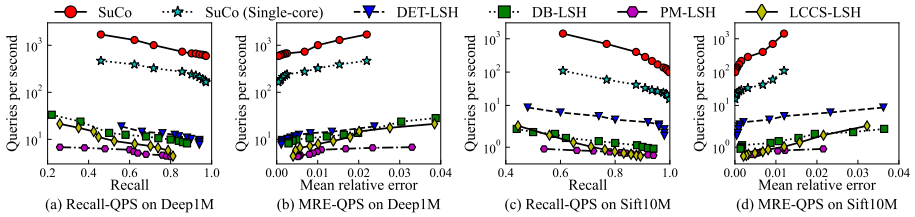


Fig. 11. Query performance comparison between SuCo and competitors that provide theoretical guarantees.

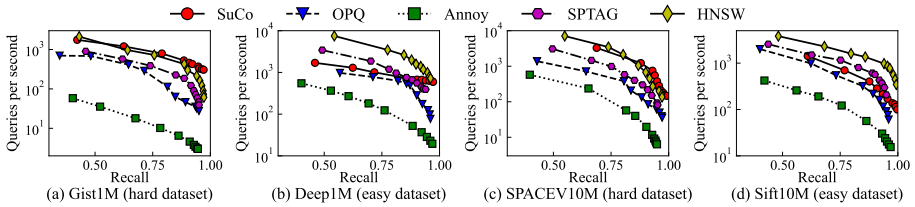


Fig. 12. Query performance comparison between SuCo and competitors that do not provide theoretical guarantees.

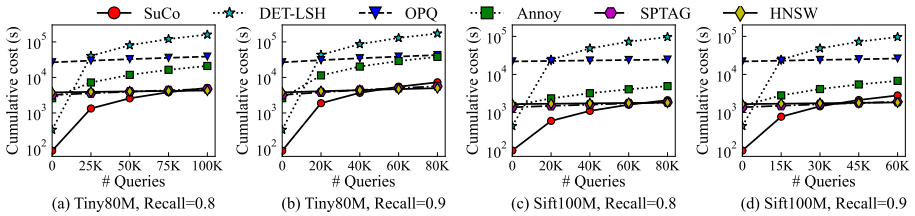


Fig. 13. Cumulative query cost (start with the indexing time); comparison to methods with/without guarantees.

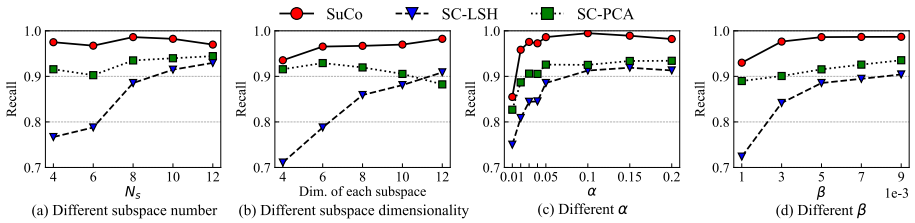


Fig. 14. Query performance of SC-based methods with different data preprocessing techniques on Sift10M.

need to perform K-means clustering of $\sqrt{K} = 50$ twice, which makes SuCo highly efficient for indexing. However, the tree structure partitions data points in a fine-grained manner, while the graph structure connects each data point to its sufficiently close neighbor points, which makes the index heavyweight and requires longer indexing time and larger memory space (Annoy, SPTAG, and HNSW). Although OPQ also uses IMI as the index structure, its query strategy requires IMI to be able to divide data points very finely (different from our subspace collision framework), so the K of K-means clustering needs to be very large ($\sqrt{K} = 2^{10} - 2^{14}$), which seriously restricts the indexing efficiency of OPQ.

Table 6. The time and space complexity of the state-of-the-art methods (for notation, see Section 5.7)

	SuCo	DET-LSH	OPQ	Annoy	SPTAG	HNSW
IT	$O(n(\sqrt{K}dt + N_s))$	$O(LKn(d + \log N_r))$	$O(n(\sqrt{K} + k^*)dt)$	$O(Lnd \log n)$	$O(nd \log n)$	$O(nd \log n)$
IS	$O(\sqrt{K}d + nN_s)$	$O(LKn)$	$O(n(m \log k^* + \log \sqrt{K}) + (\sqrt{K} + k^*)d)$	$O(Ldn)$	$O(n(d + E))$	$O(nE)$
QT	$O(n(\alpha N_s + \beta d + \log(\beta n)) + \sqrt{K}(d + \alpha\sqrt{K} \log \sqrt{K}))$	$O(n(\beta d + LK \log N_r))$	$O(n(\gamma m + \beta d) + (\sqrt{K} + k^*)d)$	$O(Ld \log n + \beta nd)$	$O(d \log n)$	$O(d \log n)$

5.5.2 Query performance. Figure 12 gives the recall-QPS curves of SuCo and competitors that do not provide theoretical guarantees. We found that SuCo and HNSW outperform other methods, and on the hard datasets (Gist1M, SPACEV10M), SuCo performs better, while on the easy datasets (Deep1M, Sift10M), HNSW performs better. Graph-based methods (such as HNSW) sacrifice indexing time and space consumption in exchange for higher query efficiency because they need to identify the near neighbors for each data point in the dataset (and connect to them) during the indexing phase, while in the query phase, they only need to search on a gradually converging path. However, on the hard datasets, HNSW's greedy query strategy is prone to falling into local optimal subgraphs and is difficult to escape. In addition, HNSW requires few and accurate query results directly from the index, so its query strategy has low fault tolerance. On the contrary, SuCo only needs to find many (about 3%-5%) but not necessarily accurate data points from its index as collisions, so its query strategy is highly fault-tolerant and more suitable for hard datasets. It is important to emphasize that SuCo has to pay the cost of providing guarantees for its answers; other competitors, that do not provide any guarantees, do not pay this cost. Even with this cost, SuCo achieves top performance (best for hard datasets) when compared to competitors that do not provide theoretical guarantees.

5.6 Overall Evaluation

We now evaluate the cumulative indexing and query time cost of methods with theoretical guarantees (SuCo and DET-LSH) and without (OPQ, Annoy, SPTAG, and HNSW). Figure 13 shows the cumulative query costs of all methods, where the cost starts with the indexing time. Compared with methods that provide theoretical guarantees (e.g., DET-LSH), SuCo always has a considerable performance advantage, ranging between 1 and more than 2 orders of magnitude. Even when compared with methods that do *not* provide theoretical guarantees (OPQ, Annoy, SPTAG, and HNSW), SuCo also has an advantage: it creates the index and answers 40K-80K queries before the best competitor (i.e., HNSW) answers its first query. Note that SuCo performs better on hard datasets: compared with the Sift100M (easy) dataset, SuCo answers more queries on the Tiny80M (hard) dataset before HNSW answers its first query.

5.7 Complexity Comparison and Analysis

Since the index structures and query strategies of the state-of-the-art methods are different (refer to Table 6), it is hard to make a direct comparison among them. In our analysis, we will focus on the main parts that affect the complexity of each method (ignoring secondary terms marked in gray in Table 6, which are not at the same complexity level as the main items, e.g., $O(\log n)$ and $O(n)$) and conduct a comparative analysis based on these parts. For all methods, n, d, β, t have the same meanings as defined in this paper.

In terms of indexing time (IT), Annoy (L is the number of trees), HNSW, and SPTAG need more than $O(n \log n)$ complexity, which is time-consuming. Although OPQ has the same complexity level $O(n)$ as SuCo and DET-LSH (L and K are the number and dimensionality of projected spaces, and N_r is the number of regions in each projected space), it requires two layers of clustering for quantization (the number of centroids is K and k^*, respectively), so it takes a long time to build the index. Therefore, SuCo and DET-LSH are suitable for scenarios sensitive to indexing time. In terms of index space (IS), SuCo and OPQ are cluster-based indexes, and their complexity coefficients N_s, m (the number of subvectors), and k^* are relatively small. The tree-based indexes (DET-LSH, Annoy), graph-based index (HNSW), and hybrid index (SPTAG) have larger complexity coefficients, i.e., the number of trees (L) and the number of neighbors for each vertex in the graph (E). Therefore, SuCo and OPQ are suitable for scenarios with limited index space. In terms of query time (QT), HNSW and SPTAG have $O(\log n)$ complexity, while other methods rely on re-ranking, with a complexity of $O(\beta nd)$. Since the subspace collision framework can effectively select candidate points, SuCo has a smaller re-rank ratio β than DET-LSH, OPQ (γ is the candidate pool ratio), and Annoy. Therefore, HNSW, SuCo, and SPTAG are suitable for scenarios sensitive to query time, as validated by the experimental results in Section 5.4 and Section 5.5.

5.8 Impact of Data Preprocessing Techniques

Our proposed subspace collision framework consists of two ideas: (1) use a simple division strategy to preprocess the raw data into subspaces and (2) use α and β to generate a pool of NN candidates and then pick the best NNs from that pool. We explored the effects of combining different data preprocessing techniques (LSH for projection, Principal Component Analysis (PCA) for dimensionality reduction, and our simple division) with the subspace collision framework, as shown in Figure 14. Figure 14 (a) and Figure 14 (b) show that SuCo outperforms SC-LSH and SC-PCA in query accuracy under different subspace settings. In addition, SuCo's data preprocessing speed is 4x and 12x faster than SC-LSH and SC-PCA, respectively. Figure 14 (c) and Figure 14 (d) explore the query performance of SC-based methods under different alpha-beta settings: SuCo exhibits the best performance. Therefore, our proposed simple division strategy is more beneficial to the subspace collision framework than previously proposed data preprocessing techniques.

6 Conclusions

In this paper, we first designed SC-score, a metric that follows the "Pareto principle" and can act as a proxy for the Euclidean distance between data points. Next, we proposed a novel ANN search framework called subspace collision, which can achieve high recall and provide theoretical guarantees on the quality of its results. Then, we proposed SuCo, which achieves efficient and accurate ANN search by designing a clustering-based lightweight index and query strategies for the subspace collision framework. Finally, we conducted extensive experiments, and the results demonstrate the superiority of SuCo in indexing and query answering performance.

In future work, we will combine deep learning to explore more efficient index structures under the subspace collision framework.

Acknowledgments

X. Lee and B. Peng partially supported by National Natural Science Foundation of China (62202450) and FUXI Institution-CASICT Interenet Infrastructure Laboratory (E051570). Z. Liao partially supported by National Natural Science Foundation of China (NSFC-62206101, NSFC-12141107), and Guangdong Provincial Key Laboratory of Mathematical Foundations for Artificial Intelligence (2023B1212010001). T. Palpanas partially supported by EU Horizon projects AI4Europe (101070000), TwinODIS (101160009), ARMADA (101168951), DataGEMS (101188416) and RECITALS (101168490).

References

[1] 2003. Bounds and Approximations for Moments of Order Statistics. In *Order Statistics*. John Wiley & Sons, Ltd, Chapter 4, 59–93. doi:10.1002/0471722162.ch4

[2] 2003. Expected Values and Moments. In *Order Statistics*. John Wiley & Sons, Ltd, Chapter 3, 33–58. doi:10.1002/0471722162.ch3

[3] Alexandr Andoni. 2005. LSH Algorithm and Implementation (E2LSH). https://web.mit.edu/andoni/www/LSH/index.html.

[4] Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 793–801.

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

[6] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. ELPIS: Graph-Based Similarity Search for Scalable Data Science. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1548–1559.

[7] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *PACMMOD* (2025).

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

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

[10] Artem Babenko and Victor Lempitsky. 2015. Tree quantization for large-scale similarity search and classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4240–4248.

[11] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2055–2063.

[12] Jonathan Bac, Evgeny M Mirkes, Alexander N Gorban, Ivan Tyukin, and Andrei Zinovyev. 2021. Scikit-dimension: a python package for intrinsic dimension estimation. *Entropy* 23, 10 (2021), 1368.

[13] Jenny A. Baglivo. 2005. *Mathematica Laboratories for Mathematical Statistics*. Society for Industrial and Applied Mathematics. doi:10.1137/1.9780898718416

[14] Erik Bernhardsson. 2015. *Approximate Nearest Neighbors in C++/Python optimized for memory usage and loading/saving to disk*. https://github.com/spotify/annoy

[15] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*. 97–104.

[16] Gunnar Blom. 1958. *Statistical Estimates and Transformed Beta-variables*. Wiley.

[17] Christian Böhm. 2000. A cost model for query processing in high dimensional data spaces. *ACM Transactions on Database Systems (TODS)* 25, 2 (2000), 129–178.

[18] Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. 1999. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. 312–321.

[19] Leonid Boytsov and Bilegsaikhan Naidan. 2013. Learning to prune in metric and non-metric spaces. *Advances in Neural Information Processing Systems* 26 (2013).

[20] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and information systems* 39, 1 (2014), 123–151.

[21] Lawrence Cayton. 2008. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th international conference on Machine learning*. 112–119.

[22] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng. 2023. Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1140–1153.

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

[24] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.

[25] Romain Couillet and Zhenyu Liao. 2022. *Random Matrix Methods for Machine Learning*. Cambridge University Press.

[26] Sanjoy Dasgupta and Yoav Freund. 2008. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 537–546.

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

[28] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*. 577–586.

[29] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2022. Hercules against data series similarity search. *Proceedings of the VLDB Endowment* 15, 10 (2022), 2005–2018.

[30] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2020. Scalable Machine Learning on High-Dimensional Vectors: From Data Series to Deep Network Embeddings. In *International Conference on Web Intelligence, Mining and Semantics WIMS*. 1–6.

[31] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2019. Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. *Proc. VLDB Endow.* 13, 3 (2019), 403–420. doi:10.14778/3368289.3368303

[32] Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and George Paterakis. 2023. FreSh: A Lock-Free Data Series Index. In *42nd International Symposium on Reliable Distributed Systems, SRDS*. IEEE, 209–220. doi:10.1109/SRDS60354.2023.00029

[33] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2021), 4139–4150.

[34] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474.

[35] Keinosuke Fukunaga and Patrenahalli M. Narendra. 1975. A branch and bound algorithm for computing k-nearest neighbors. *IEEE transactions on computers* 100, 7 (1975), 750–753.

[36] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 541–552.

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

[38] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.

[39] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[40] Robert Gray. 1984. Vector quantization. *IEEE Assp Magazine* 1, 2 (1984), 4–29.

[41] Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE transactions on information theory* 44, 6 (1998), 2325–2383.

[42] Alexander Hinneburg, Charu C Aggarwal, and Daniel A Keim. 2000. What is the nearest neighbor in high dimensional spaces?. In *26th Internat. Conference on Very Large Databases*. 506–515.

[43] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[44] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).

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

[46] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 861–864.

[47] Kerstin Johnsson, Charlotte Soneson, and Magnus Fontes. 2014. Low bias local intrinsic dimension estimation from expected simplex skewness. *IEEE transactions on pattern analysis and machine intelligence* 37, 1 (2014), 196–202.

[48] Yannis Kalantidis and Yannis Avrithis. 2014. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2321–2328.

[49] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906* (2020).

[50] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: A Scalable Bottom-Up Approach for Building Data Series Indexes. *Proceedings of the VLDB Endowment* 11, 6 (2018).

[51] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2019. Coconut: sortable summarizations for scalable indexes over static and streaming data series. *VLDB J.* 28, 6 (2019), 847–869.

[52] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony KH Tung. 2020. Locality-sensitive hashing scheme based on longest circular co-substring. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2589–2599.

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

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

[55] Ying Li, Jiuqi Wei, Ziyu Fei, Yufan Fu, and Xiaodong Lee. 2024. DiSAuth: A DNS-based secure authorization framework for protecting data decoupled from applications. *Computer Networks* 254 (2024), 110774.

[56] Zhenyu Liao, Romain Couillet, and Michael W. Mahoney. 2020. A Random Matrix Analysis of Random Fourier Features: Beyond the Gaussian Kernel, a Precise Phase Transition, and the Corresponding Double Descent. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 13939—13950.

[57] Michele Linardi and Themis Palpanas. 2018. Scalable, variable-length similarity search in data series: The ULISSE approach. *Proceedings of the VLDB Endowment* 11, 13 (2018), 2236–2248.

[58] Michele Linardi and Themis Palpanas. 2020. Scalable data series subsequence matching with ULISSE. *VLDB J.* 29, 6 (2020), 1449–1474.

[59] 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. *The VLDB Journal* 30 (2021), 215–235.

[60] Yingfan Liu, Hong Cheng, and Jiangtao Cui. 2017. PQBF: i/o-efficient approximate nearest neighbor search by product quantization. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 667–676.

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

[62] Cosme Louart, Zhenyu Liao, and Romain Couillet. 2018. A Random Matrix Approach to Neural Networks. *Annals of Applied Probability* 28, 2 (2018), 1190–1248. doi:10.1214/17-aap1328

[63] Kejing Lu and Mineichi Kudo. 2020. R2LSH: A nearest neighbor search scheme based on two-dimensional projected spaces. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1045–1056.

[64] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 15, 2 (2021), 246–258.

[65] Kejing Lu, Hongya Wang, Wei Wang, and Mineichi Kudo. 2020. VHP: approximate nearest neighbor search via virtual hypersphere partitioning. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1443–1455.

[66] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[67] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[68] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. 2018. A survey of product quantization. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 2–10.

[69] Yusuke Matsui, Toshihiko Yamasaki, and Kiyoharu Aizawa. 2015. Pqtable: Fast exact asymmetric distance neighbor search for product quantization using hash tables. In *Proceedings of the IEEE International Conference on Computer Vision*. 1940–1948.

[70] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence* 36, 11 (2014), 2227–2240.

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

[72] Riley Murray, James Demmel, Michael W. Mahoney, N. Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michał Dereziński, Miles E. Lopes, Tianyu Liang, Hengrui Luo, and Jack Dongarra. 2023. Randomized Numerical Linear Algebra : A Perspective on the Field With an Eye to Software. doi:10.48550/arXiv.2302.11474 arXiv:2302.11474

[73] Gonzalo Navarro. 2002. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11 (2002), 28–46.

[74] Mohammad Norouzi and David J Fleet. 2013. Cartesian k-means. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*. 3017–3024.

[75] Themis Palpanas. 2015. Data Series Management: The Road to Big Sequence Analytics. *SIGMOD Record* (2015).

[76] Themis Palpanas and Volker Beckmann. 48(3), 2019. Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA). *SIGREC* (48(3), 2019).

[77] Marco Patella and Paolo Ciaccia. 2008. The many facets of approximate similarity search. In *First International Workshop on Similarity Search and Applications (sisap 2008)*. IEEE, 10–21.

[78] Marco Patella and Paolo Ciaccia. 2009. Approximate similarity search: A multi-faceted problem. *Journal of Discrete Algorithms* 7, 1 (2009), 36–48.

[79] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2018. ParIS: The Next Destination for Fast Data Series Indexing and Query Answering. *IEEE BigData* (2018).

[80] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2020. Messi: In-memory data series indexing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 337–348.

[81] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2020. Paris+: Data series indexing on multi-core architectures. *IEEE Transactions on Knowledge and Data Engineering* 33, 5 (2020), 2151–2164.

[82] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. Fast data series indexing for in-memory data. *VLDB J.* 30, 6 (2021).

[83] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. SING: Sequence Indexing Using GPUs. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1883–1888.

[84] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[85] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*. Springer, 291–324.

[86] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.

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

[88] Yukihiro Tagami. 2017. Annexml: Approximate nearest neighbor search for extreme multi-label classification. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 455–464.

[89] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2009. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 563–576.

[90] Yao Tian, Ziyang Yue, Ruiyuan Zhang, Xi Zhao, Bolong Zheng, and Xiaofang Zhou. 2023. Approximate Nearest Neighbor Search in High Dimensional Vector Databases: Current Research and Future Directions. *IEEE Data Engineering Bulletin* 47, 3 (2023).

[91] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2023. DB-LSH 2.0: Locality-Sensitive Hashing With Query-Based Dynamic Bucketing. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[92] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978.

[93] Qitong Wang, Ioana Ileana, and Themis Palpanas. 2025. LeaFi: Data Series Indexes on Steroids with Learned Filters. *Proc. ACM Manag. Data* (2025).

[94] Qitong Wang and Themis Palpanas. 2021. Deep learning embeddings for data series similarity search. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 1708–1716.

[95] Qitong Wang and Themis Palpanas. 2023. SEAnet: A Deep Learning Architecture for Data Series Similarity Search. *IEEE Trans. Knowl. Data Eng.* 35, 12 (2023), 12972–12986.

[96] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. 2013. A data-adaptive and dynamic segmentation index for whole matching on time series. *VLDB* (2013).

[97] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph- and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions. *IEEE Data Eng. Bull.* 47, 3 (2023), 3–21.

[98] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. ***Steiner***-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. *PVLDB* (2024).

[99] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A Compact and Adaptive Index for Large Data Series Collections. *Proc. ACM Manag. Data* 1, 1 (2023), 111:1–111:27.

[100] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2024. DumpyOS: A data-adaptive multi-ary index for scalable data series similarity search. *The VLDB Journal* (2024), 1–25.

[101] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, Vol. 98. 194–205.

[102] Jiuqi Wei. 2024. Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. https://github.com/WeiJiuQi/SuCo.

[103] Jiuqi Wei, Ying Li, Yufan Fu, Youyi Zhang, and Xiaodong Li. 2023. Data Interoperating Architecture (DIA): Decoupling Data and Applications to Give Back Your Data Ownership. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 438–447.

[104] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment* 17, 9

(2024), 2241–2254.

[105] Yan Xia, Kaiming He, Fang Wen, and Jian Sun. 2013. Joint inverted indexing. In *Proceedings of the IEEE International Conference on Computer Vision*. 3416–3423.

[106] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2017. DPiSAX: Massively Distributed Partitioned iSAX. In *IEEE International Conference on Data Mining, ICDM*. 1135–1140.

[107] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2020. Massively Distributed Time Series Indexing and Querying. *IEEE Trans. Knowl. Data Eng.* 32, 1 (2020), 108–120.

[108] Peter N Yianilos. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Soda*, Vol. 93. 311–21.

[109] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite quantization for approximate nearest neighbor search. In *International Conference on Machine Learning*. PMLR, 838–846.

[110] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1979–1991.

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

[112] Yuxin Zheng, Qi Guo, Anthony KH Tung, and Sai Wu. 2016. Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. In *Proceedings of the 2016 International Conference on Management of Data*. 2023–2037.

[113] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for interactive exploration of big data series. In *International Conference on Management of Data, SIGMOD*.

[114] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2016. ADS: the adaptive data series index. *The VLDB Journal* 25 (2016), 843–866.

Received July 2024; revised September 2024; accepted November 2024