Tribase: A Vector Data Query Engine for Reliable and Lossless Pruning Compression using Triangle Inequalities

QIAN XU, Renmin University of China, China
JUAN YANG, Tsinghua University, China
FENG ZHANG*, Renmin University of China, China
JUNDA PAN, Renmin University of China, China
KANG CHEN, Tsinghua University, China
YOUREN SHEN, Beijing HaiZhi XingTu Technology Co., Ltd., China
AMELIE CHI ZHOU, Hong Kong Baptist University, China
XIAOYONG DU, Renmin University of China, China

Approximate Nearest Neighbor Search (ANNS) is a critical problem in vector databases. Cluster-based index is utilized to narrow the search scope of ANNS, thereby accelerating the search process. Due to its scalability, it is widely employed in real-world vector search systems. However, existing cluster-based indexes often suffer from coarse granularity, requiring query vectors to compute distances with vectors of varying quality, thus increasing query complexity. Existing work aim to represent vectors with minimal cost, such as using product quantization (PQ) or linear transformations, to speed up ANNS. However, these approaches do not address the coarse granularity inherent in cluster-based index. In this paper, we present an efficient vector data query engine to enhance the granularity of cluster-based index by carefully subdividing clusters using diverse distance metrics. Building on this refined index, we introduce techniques that leverage triangle inequalities to develop highly optimized and distinct search strategies for clusters and vectors of varying qualities, thereby reducing the overhead of ANNS. Extensive experiments demonstrate that our method significantly outperforms existing in-memory cluster-based indexing algorithms, achieving up to an impressive 10× speedup and a pruning ratio exceeding 99.4%.

CCS Concepts: • **Computing methodologies**;

Additional Key Words and Phrases: Vector Data Query Engine, Triangle Inequalities, Pruning Compression

ACM Reference Format:
Qian Xu, Juan Yang, Feng Zhang, Junda Pan, Kang Chen, Youren Shen, Amelie Chi Zhou, and Xiaoyong Du. 2025. Tribase: A Vector Data Query Engine for Reliable and Lossless Pruning Compression using Triangle Inequalities. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 82 (February 2025), 28 pages. https://doi.org/10.1145/3709743

*Feng Zhang is the corresponding author of this paper.

Authors' Contact Information: Qian Xu, Renmin University of China, China, xuqianmamba@gmail.com; Juan Yang, Tsinghua University, China, yangjuan@stargraph.cn; Feng Zhang, Renmin University of China, China, fengzhang@ruc.edu.cn; Junda Pan, Renmin University of China, China, xm_jarden@ruc.edu.cn; Kang Chen, Tsinghua University, China, chenkang@tsinghua.edu.cn; Youren Shen, Beijing HaiZhi XingTu Technology Co., Ltd., China, shenyouren@stargraph.cn; Amelie Chi Zhou, Hong Kong Baptist University, China, amelie.czhou@gmail.com; Xiaoyong Du, Renmin University of China, China, duyong@ruc.edu.cn.

ACM 2836-6573/2025/2-ART82
https://doi.org/10.1145/3709743

1 Introduction

Recent vector databases achieve efficient search through Approximate Nearest Neighbor Search (ANNS) [23]. As cluster-based index exhibits excellent query precision and provides relatively straightforward support for vector updates[1] [74], it is widely adopted in practical systems [8, 17, 26, 65, 70]. However, cluster-based methods exhibit a significant increase in time consumption during high-precision ANNS. Previous work has analyzed that computing the distance between vectors in cluster-based index consumes most of the ANNS time [21]. Therefore, efficiently pruning the distance calculation process becomes an important issue to speed up the in-memory ANNS. We find that triangle inequalities are commonly used to estimate distances in $\mathbb{R}^2$ and are utilized in many studies due to their accuracy and robustness [14, 68, 69]. Additionally, some pioneering works [51, 67] were the first to explore incorporating triangle inequalities into ANNS. However, due to coarse granularity of existing cluster-based index and the curse of dimensionality [4, 22, 55, 57], there is a need to improve pruning effectiveness from two perspectives: refining the granularity of the cluster-based index and applying triangle inequalities with multiple metrics, such as angles and distances. Therefore, our work focuses primarily on proposing a finer-grained index and utilizing diverse pruning strategies.

Pruning the vector search process using triangle inequalities yields three compelling advantages. First, previous work has shown that computing the distances between the vectors takes up the vast majority of the time in ANNS. In our experiments, the vector distance computation accounts for 99.7% of the time when performing searches with 99% precision on the *msong* dataset, and optimizing this time overhead will greatly speed up ANNS. Second, due to the wide distribution of vectors, the neighborhood of a query vector usually contains only a small number of vectors. Using triangle inequalities to perform plausible vector pruning does not affect the accuracy of the final result. Third, many existing works focus on quickly locating the neighbors of vectors [18, 21, 37, 45] or on combining with hardware for faster distance computation [43, 53], but do not focus on pruning low-quality distance computation. Due to the simplicity of the triangle inequalities logic, we can integrate it with these works. For example, we combine our pruning well with Faiss [17], one of the best open-source in-memory vector search libraries, and obtain up to 10× speedup.

There are many cluster-based index works. Today's cluster-based indexes [3, 5, 8, 17, 17, 21, 31–33, 74, 77] first compute the distance between the query vector and the center of each cluster. They then select a number of clusters whose centers are closest to the query vector and compute the distance between the vectors in these clusters and the query vector, maintaining the results in a max heap. We refer to this method as full clustering search. The time complexity of full clustering search is $O(D{\cdot}|C_{num}|{\cdot}|C|)$, where D is the dimension of the vector, $|C_{num}|$ is the number of selected clusters, and $|C|$ is the average number of vectors in each cluster. While cluster-based index vector databases can reduce the number of vectors queried during searches, they face two main problems. First, the granularity of existing cluster-based indexes is too coarse; they cannot further differentiate and filter elements within the same cluster. Second, the conditions for pruning cannot be efficiently updated. As search results are continuously optimized during vector computation, it becomes increasingly difficult for new vectors to replace existing results. However, in today's cluster-based searching, each selected vector retains the same status throughout the search process.

Utilizing the triangle inequality to accelerate ANNS presents two key challenges. First, due to the "curse of dimensionality" (discussed in Section 3.2), the distribution of vectors in high-dimensional spaces is much sparser compared to low-dimensional spaces, which affects the pruning effectiveness of the triangle inequalities. Second, since the triangle inequalities introduce computational overhead, and many ANNS methods support hardware acceleration of distance calculations, this overhead is

[1]In contrast, graph-based index may need to reconstruct the graph.

further amplified. Therefore, the challenge lies in how to leverage the triangle inequality property for ANNS pruning with minimum additional computational cost.

We have made two key observations. First, for a specified query vector, vectors located in different clusters should have different search statuses. Vectors in clusters closer to the query point are more likely to be part of the final search result. Even vectors within the same cluster will have different pruning conditions due to their varying positions within the cluster. However, the granularity of the current cluster-based index is too coarse. Each vector in the selected clusters is treated with equal status to the query vector, regardless of the cluster they belong to or their position within the cluster. Second, for vector queries, regardless of the vector's dimension, triangle inequalities can be derived based on both distance and angle. From a distance perspective, the distance from A to vector B is never greater than the distance from vector A to C plus the distance from vector C to B. From an angle perspective, the three vectors $\overrightarrow{OA}$, $\overrightarrow{OB}$, and $\overrightarrow{OC}$ and their angles in the high-dimensional space always satisfy $\angle AOB + \angle BOC < \angle AOC$.

Based on these two observations, we have designed **Tribase**, a new vector data query engine for lossless pruning based on triangle inequalities. Tribase introduces three main innovations. First, Tribase leverages the similarity of vectors within a single cluster to perform fine-grained, multi-level partitioning of the vectors. Compared to simply dividing vectors by clusters, this approach obtains more information that helps characterize the vector distribution. Second, Tribase is capable of leveraging angle-based triangle inequalities in vector queries. This capability introduces new opportunities for exploring triangle inequality pruning based on the proposed fine-grained index, further enhancing the efficiency of vector search. Third, Tribase can meet different indexing costs and pruning effects, thanks to the flexible combination of fine-grained pruning we provide and the scaling of the triangle inequalities to adjust the pruning intensity.

We evaluate Tribase against the state-of-the-art (SOTA) solution Faiss [17] using ten datasets generated from various types of real data. Additionally, we have integrated the pruning modules of Tribase into Faiss [17] and compared it with the SOTA pruning solution ADSampling [21] to validate the scalability of Tribase. Experiments show that Tribase can prune vectors in ANNS by up to 99.4% and achieve a speedup of up to 10× compared to Faiss.

We summarize the major contributions of this paper as follows:

- We identified that the coarse granularity of current cluster-based indexes results in unnecessary distance computations. To address this, we propose a novel method for fine-grained indexing construction, which improves pruning efficiency by reducing the granularity of cluster-based indexes and optimizing distance computations.
- We design and implement Tribase, an efficient vector query engine to utilize both distance- and angle-based triangle inequalities in a fine-grained index for ANNS pruning. We also integrate the pruning modules of Tribase into Faiss.
- We conduct comprehensive experiments demonstrating that, in comparison to the open-source system Faiss, Tribase reduces computations by up to 99.4%, achieving average and maximum speed-ups of 3.11× and 10×, respectively.

2 Background and related work

2.1 Vector database systems

With the development of deep learning, vectors are now utilized in nearly all applications, such as search engines [8, 9, 25, 35, 40, 73], e-commerce platforms [39], recommendation systems [34, 38, 41, 72] and natural language processing tasks [47, 54]. Vectors are employed to represent a wide variety of data, including videos, texts, images, and audio. Efficient vector search is essential for

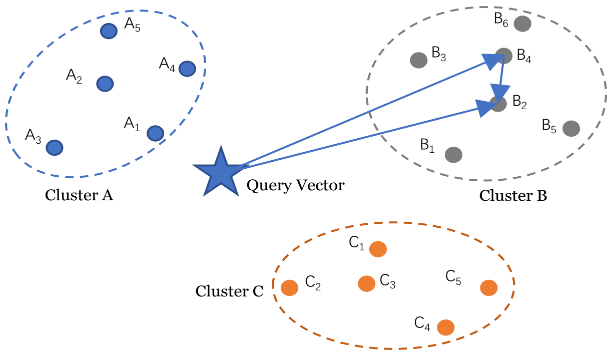


Fig. 1. An example of search in cluster-based index.

enhancing the performance of these applications, necessitating the construction of efficient vector databases [19, 45, 53, 74–76].

Recent vector database systems face significant challenges. On one hand, the scale of vector data is rapidly increasing. For instance, during Alibaba's shopping festival, 500PB of data is generated daily [20], while YouTube receives over 500 hours of content every minute [2]. At the same time, each vector may have hundreds of dimensions [46, 52, 58]. For instance, GPT-3 utilizes a 12288-dimensional embedding layer [7]. Additionally, according to the UCI Machine Learning Repository, a well-known source for machine learning datasets [1], there are over 100 open-source datasets with dimensions exceeding 100. On the other hand, the "online" nature of these services [12, 24, 29, 50] necessitates vector searches be completed within milliseconds. This strict latency requirement conflicts with the exact search algorithms used by relational databases [10], as almost all vectors need to be scanned if stored in these databases.

To quickly retrieve the vectors satisfying the requirements, vector database systems use indexes to obtain the addresses of the corresponding vectors. There are mainly two types of indexes used: graph-based indexes and partition-based indexes (which include cluster-based [3, 5, 8, 17, 31–33, 74, 77], hash-based [11, 28, 56, 66, 71], and high-dimensional tree-based [6, 42, 49, 64] methods). Graph-based index methods [13, 16, 19, 20, 27, 45, 61–63, 76] construct a graph based on the distances between vectors, where the query process is equivalent to a breadth-first search on the constructed graph. Partition-based indexes, on the other hand, divide the vectors into partitions according to specific criteria. For example, cluster-based index methods build clusters based on the distances between vectors. The query process first identifies the nearest clusters and then examines the vectors within each cluster. There are other works that accelerate ANNS from a hardware perspective, such as CPU-centric [19, 21], RAM-centric [8], and disk-centric [8, 30], which is orthogonal to building high-quality indices.

2.2 Triangle inequalities in high dimensions

Triangle inequalities are a property used in machine learning and similarity search for optimizing algorithms and reducing computation [15, 51, 59, 67]. Due to their simple form and universally valid conclusion, triangle inequalities have a wide range of applications [14, 68, 69]. Interestingly, we find that they can also be applied to the vector search problem, providing lossless *distance* and *angle* pruning during the search process. For example, as shown in Figure 1, in the triangle formed by the query point, B_2, and B_4, the distance from B_2 to the query point Q always satisfies the property $\|\overrightarrow{QB_4} - \overrightarrow{B_2B_4}\|_p \leq \|\overrightarrow{QB_2}\|_p \leq \|\overrightarrow{QB_4} + \overrightarrow{B_2B_4}\|_p$. This means, if $\|\overrightarrow{QB_4} - \overrightarrow{B_2B_4}\|_p$ is longer than any element currently in the top-K set during the search, we can safely prune out B_4 and reduce the query time. Due to the high dimensionality of vector data, we first provide the proof of triangle inequalities for distance and angle in $\mathbb{R}^P$ for $P \geq 2$.

Triangle inequalities for distance. The distance-based triangle inequalities can be expressed as $\|x + y\|_p \leq \|x\|_p + \|y\|_p$, where x and y are two vectors. The proof is in Appendix B.

Triangle inequality for angle. The angle-based triangle inequalities can be defined as follows: let O, A, B, C be four points in a vector space, and we have $\angle AOC \leq \angle AOB + \angle BOC$. The proof is in Appendix C.

3 Motivation and challenges

3.1 Motivation

Motivation1: Coarse granularity of existing cluster-based index. Cluster-based index vector systems partition vectors into clusters and determine whether to compute the distance between the query vector and each vector in a cluster based on the distance between the query vector and the center of the cluster at query time. There are two optimization opportunities for cluster-based indexing.

First, the current cluster-based index employs only two classifications for clusters: “accept” or “reject”. However, based on the distance from the cluster center to the query point, we can set different acceptance levels for these clusters. Figure 1 illustrates a query vector and three nearest clusters requiring distance computation. For this query vector, the center of cluster B is farther from the query vector than the center of cluster A, so the acceptance level of cluster A is higher than that of cluster B.

Second, even points within the same cluster should have different acceptance levels due to varying directions and distances from the cluster center. As shown in Figure 1, when calculating the distances from the vectors in cluster A to the query point, vector A_1 has a higher acceptance level because it is in a part of the cluster that is closer to the query point compared to vector A_4. Therefore, the large clustering granularity of current cluster-based index and the lack of additional description fields lead to the introduction of too many query vectors that do not need to be processed during the query. This paper addresses and improves upon this deficiency.

Motivation 2: Potential of triangle inequalities on ANNS pruning. Obtaining high-precision topK (typically a small integer less than 1000) nearest neighbors for the query vector requires distance calculations between millions or even billions of vectors. This process consumes almost all of the ANNS time. In our experiments, for most datasets, the distance calculations took over 98% of the time when searching with an accuracy exceeding 99%. This aligns with the experiments conducted by [21]. Among the existing systems designed to reduce the overhead of distance computations [17, 21, 76], dimension reduction methods such as PQ [17, 33] or LSH [11, 35, 48, 60] cannot ensure lossless compression. Methods based on early termination, such as VBASE [76], cannot guarantee the accuracy of the results at the end of the search. Sampling-based methods [21, 36], such as ADSampling [21], cannot ensure the precision of pruning, leading to an inability to achieve lossless pruning. Therefore, *efficient pruning that guarantees lossless results remains an unresolved challenge.* The triangle inequalities, due to their lossless nature and controllable pruning strength, have become an important method for addressing this challenge.

3.2 Challenges

Although the triangle inequalities property has the potential to reduce ANNS latency, it is not trivial to design pruning techniques based on this property due to the following reasons.

Challenge 1: Curse of dimensionality. The term "curse of dimensionality" [4, 22, 55, 57] describes the fact that, when the dimension of data increases, the volume of solution space grows exponentially, leading to data sparsity and making data analysis increasingly difficult. When talking about ANNS, the curse of dimensionality has two types of impact when comparing high-dimensional vectors.

- *Impact on distance calculation.* Most of the points in a sphere in $\mathbb{R}^P$ are distributed on the shell in high-dimensional spaces.
- *Impact on angle calculation.* Two random vectors in high dimensional spaces are almost always perpendicular.

The examples of these two impacts are in Appendix D.

Challenge 2: Combination of triangle inequalities and fast ANNS. With the rapid advancement of hardware, the time required to compute the distance between two vectors in high-dimensional space is decreasing [53, 75]. However, the use of triangle inequalities introduces additional distance computations that are otherwise unnecessary in traditional ANNS calculations. For example, in order to prune distance calculation using triangle inequalities, we have to calculate fields related to triangle inequalities, which requires more than 10× (detailed in Section 4.2) additional computation time at runtime. Thus, the challenge lies in modifying existing index structures to incorporate these additional distances quickly and with minimal overhead, thereby mitigating the negative impact of the triangle inequalities on ANNS.

4 System design

4.1 Overview

To address the current shortcomings of cluster-based index and avoid the curse of dimensionality, we develop Tribase, a vector search engine that efficiently reduces the time of ANNS in high-dimensional spaces by utilizing triangle inequalities shown in Figure 2. Like other search engines [8, 17, 30], Tribase has the independent capability to indexing and perform approximate searches on vectors. However, different from other ANNS engines [8, 17], Tribase leverages the information in the cluster-based index to accelerate the query process.

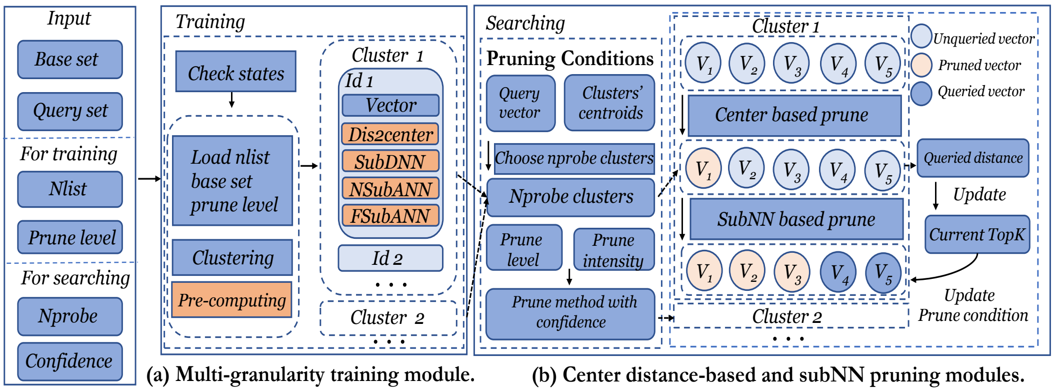


Fig. 2. Tribase overview.

Solution to challenges. Tribase strives from three aspects to address the above mentioned challenges.

First, to address the curse of dimensionality on *distance calculation*, Tribase proposes to pinpoint as many small pruning spaces as possible in clusters to improve the efficacy of pruning. Specifically, in addition to pruning based on the cluster center, we explore the option of pruning vectors at the cluster edges by examining their neighboring points, thereby alleviating the issue of requiring extremely large pruning radius.

Second, to address the curse of dimensionality on *angle calculation*, Tribase represents vectors within clusters as their residuals from the cluster center. This approach ensures vector direction diversity while using the cosine rule to reduce the computational cost of angle calculations, thereby mitigating the issue of dimensionality in angle computations.

Third, to minimize the additional computational overhead introduced by the new fields required by triangle inequalities, Tribase designs a fine-grained indexing method that allows precomputation of subsequent pruning fields within user-controllable time and space overheads, providing different levels of index quality based on the granularity of the precomputation. This can effectively mitigate the issue of introducing unnecessary additional field calculations during the query phase.

Modules. Tribase contains three main modules to incorporate the above mentioned techniques, as shown in Figure 2. Specifically, the *multi-granularity training module* trains the raw vectors at different granularity levels to obtain the distances and angles required for triangle inequality pruning. This is done offline time, in order to save the additional computation cost of triangle inequalities during the query phase. Tribase uses the other two modules, namely the *center distance-based pruning module* and *subNN pruning module*, to implement the pruning logic based on triangle inequalities. The center distance-based pruning module performs coarse-grained pruning at query time based on the distances between query vectors, cluster centers, and vectors within clusters. The subNN pruning module, which includes subNN distance-based pruning and subNN angle-based pruning, further performs fine-grained pruning on the vectors that have undergone coarse pruning.

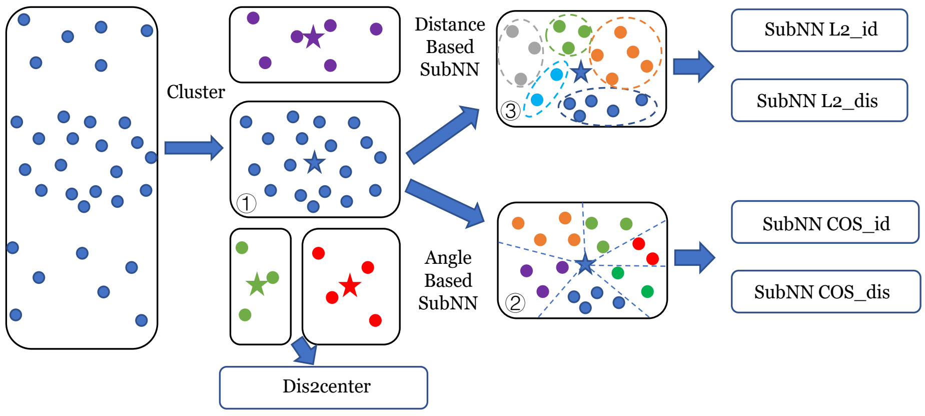
Novelties. Tribase introduces several novel aspects. First, Tribase enhances the training process, demonstrating that additional useful information can be obtained during training, beyond simply dividing the vectors according to their distribution. Second, Tribase performs pruning based on both distance- and angle-based triangle inequalities during vector queries, ensuring the correctness of these inequalities through theoretical proof, which guarantees the reliability of the pruning process. To the best of our knowledge, no previous work applies both the angle-based and multi-grained distance-based triangle inequalities to ANNS pruning. Finally, by indexing and pruning the index of Tribase with different modules and intensities, Tribase can meet different indexing and pruning effects.

4.2 Multi-granularity training module

We next introduce the multi-granularity training module of Tribase.

Design. The primary purpose of this module is to pre-compute the lengths and angles used in the triangle inequalities. Without knowing the query vectors, we can only pre-save the distances between each vector and the center of its cluster, as well as the distances between each vector and its nearest neighbor in terms of angle and distance. To compute the necessary fields for subsequent pruning with minimal overhead, Tribase retains useful information from the original indexing process and adds a new search module to capture information not available from the initial clusters. Tribase allows users to make trade-offs between preprocessing overhead and pruning effectiveness. High-quality fields require higher preprocessing costs but offer better pruning performance.

Example. We show an example of Tribase's multi-granularity training module in Figure 3. The first step in this module is to divide the vectors into clusters, similar to normal cluster-based training. However, while assigning each vector to the nearest cluster center, Tribase also records the distance of each vector to its cluster center as the first field after dividing the clusters (labeled



(a)Clustering and preserving dis2center. (b)SubNN searching and preserve subNN L2 and cos.

Fig. 3. An example of Tribase's training module.

in blue in Figure 3 (a)). Next, Tribase performs a subNN search for the vectors inside each cluster to obtain the K-nearest neighbors of each vector in the same cluster based on angle and distance (labeled in green, yellow, and grey in Figure 3 (b)). These fields are then stored along with the previously computed distances between the vector and its cluster center, as well as the vectors themselves.

Advantages. Performing an additional subNN search offers three advantages over the traditional training process. First, by using the subNN search, Tribase successfully segments the widely distributed spherical shell in high-dimensional space into many smaller regions. Second, given that the vectors themselves are high-dimensional and occupy a larger portion of the space, Tribase's subNN search incurs minimal additional space overhead by controlling the value of K. Third, since the subNN search is conducted during the training phase, it does not increase the computational overhead during queries.

Time complexity. The total time complexity of Tribase consists of two parts: 1) clustering all nodes and computing the distance between each vector and its clustering center, and 2) performing subNN queries for vectors in each cluster. Both Tribase and the state-of-the-art Faiss [17] use K-means as the clustering method in large-scale experiments.

The first step involves clustering all nodes and computing the distance between each vector and all clusters. Like Faiss, Tribase samples a portion of the vectors and performs K-means on these sampled vectors to determine the clustering centers. The time complexity of this step is $O(I{\cdot}S{\cdot}K{\cdot}d)$, where I is the number of iterations, S is the number of sampled vectors, K is the number of clustering centers, and d is the dimension of the vectors. After obtaining all the clustering centers, both Tribase and Faiss assign all vectors to the nearest clustering centers, which requires calculating the distance between all vectors and each clustering center. The complexity of this step is $O(N{\cdot}K{\cdot}d)$, where N is the total number of vectors. Since the distance between each vector and its clustering center is naturally available at this step, we just need to save it directly, incurring no additional time overhead.

In Tribase's subNN search process, to get the K-nearest neighbors in the same cluster based on angle and distance, Tribase performs an intra-cluster global search to find the top-K nearest neighbors within each cluster. The complexity of this intra-cluster search is $O(K{\cdot}(N/K)^2{\cdot}d)$, and the complexity of maintaining the optimal *subK* results is $O(K{\cdot}(N/K){\cdot}log(subK))$. Thus, the complexity of this step is $O((N^2/K){\cdot}d + N{\cdot}log(subK))$.

In summary, the total time complexity of Tribase is $O((I{\cdot}S + N){\cdot}K{\cdot}d + (N^2/K){\cdot}d + N{\cdot}log(subK))$. Compared to Faiss, Tribase requires an additional expenditure of $O((N^2/K){\cdot}d + N{\cdot}log(subK))$.

Taking I=20, S=10000, N=1000000, K=1000, $subK$=10, d=200, we estimate that the time cost of the indexing phase of Tribase is about 2.19 times that of Faiss. Note that opting for slightly lower-quality indexes can significantly reduce the training phase's time overhead, with minimal impact on search performance. Please refer to the experiments in Sections 6.4.2 and 6.4.3.

It is worth mentioning that performing SubNN search in advance can greatly reduce the time of subsequent ANNS. For example, suppose we need to query the nearest neighbors for $|Q| = 10^3$ query points. The dataset consists of $K = 10^3$ clusters containing a total of $N = 10^6$ vectors. If we aim to perform distance computations for 1/10 of the points, this results in $N/10 \cdot |Q| = 10^8$ distance calculations. In Tribase, calculating the distance or angle for each point within the relevant clusters requires $N \cdot (N/K) = 10^9$ distance calculations. This is 10× the original KNN query.

Space complexity. The space can be divided into two parts: the space required for storing the nodes themselves, which is $N \cdot d$, and the additional space introduced by the SubNN index, which is $N \cdot d \cdot subK$. Since high-dimensional vectors themselves require a large amount of storage space, the additional attributes stored by Tribase are relatively small in comparison. For example, assuming a vector dimension d=200, storing the fields required for pruning in the center distance-based pruning module would increase the space by just 0.49%. If $subK$=5, storing the fields required for the subNN distance-based pruning module would increase the space by 4.98%, while storing the fields required for the subNN angle-based pruning module would increase the space by 9.95%. We present experimental results demonstrating the space overhead introduced by the SubNN index in Section 6.5.1.

Table 1. Sign and description.

Sign	Description
Q	Query vector
C_i	ith centorid
R_{C_i}	$\max_{x \in C_i} \|x - c_i\|$
D_{ji}	$\|\mathbf{v}_j - \mathbf{c}_i\|$
P	Current select vector
$C_{[P]}$	$\mathbf{c}_i = \text{centroid}(C_i)$ where $\mathbf{P} \in C_i$
$TopK\_W$	Worst result in the current TopK set
$B(A, D)$	$\{x \in \mathbb{R}^n : \|x - A\| < D\}$

Compatibility with update. Existing research has already demonstrated the inherent advantages of cluster-based index during updates [74], which only require reclustering the vectors within the affected clusters. Tribase introduces two additional fields that need updating during this process. The first is the distance of each vector to its cluster center, which is naturally updated during the reclustering. The second is the subNN index can perform a localized search within the new clusters after reclustering. Since this process essentially involves partially reconstructing the cluster-based index, the proportion of time subNN spends on updates is comparable to the time spent during the initial index construction.

4.3 Center distance-based pruning module

Next, we introduce the center distance-based pruning module and the subNN pruning module of Tribase. We show the pruning modules of Tribase in Algorithm 1. The center distance-based pruning module is responsible for coarse-grained pruning, while the subNN pruning module handles fine-grained pruning based on the results from the center distance-based pruning module. The symbols and their descriptions used in this context are defined in Table 1. For a query vector Q and a set of cluster centers C_1,C_2...C_n, similar to other cluster-based indexes, Tribase first calculates the distance between Q and C_1,C_2...C_n and then selects the optimal $nprobe$ clusters specified by the user for further searching.

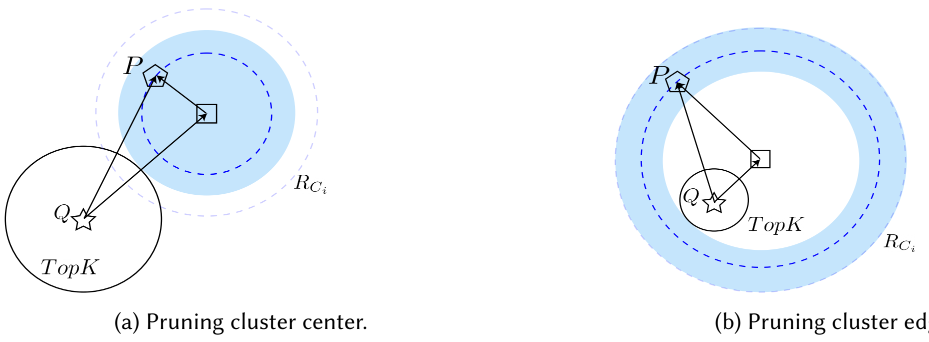
Algorithm 1: Tribase Pruning

```
Function TriPruning($Q$, Centroids $C, K$):
    $T \leftarrow$ create_heap($K$)
    for $Cp \in C$ do
        $Ip \leftarrow$ bitmap($|Cp|$)                // Initialize a bitmap for the condition if pruned
        for $P \in Cp$ do
            if $Ip[P]$ and not CenterDistPrune($Cp, Q, P$) then
                $Dpq \leftarrow$ dist($P, Q$)
                $Nnd \leftarrow \mathrm{NN}_d(P)$, $Nnc \leftarrow \mathrm{NN}_a(P)$
                $Ip \leftarrow$ update SubNNDistPrune($Nnd, Dpq$),
                    update SubNNAngPrune($Nnc, Dpq$)
Function center_distance_pruning($C_{[P]}, Q, TopK\_W$):
    if $P$ in region satisfy triangle inequalities then
        prune P
Function subnn_distance_pruning($Q.NNd$, dist($P, Q$)):
    for $p \in Q.NNd$ do
        if $p$ in region satisfy triangle inequalities then
            $Ip[P] \leftarrow$ false
Function subnn_angle_pruning($Q.NNa$, dist($P, Q$)):
    for $p \in Q.NNa$ do
        if $p$ in region satisfy triangle inequalities then
            $Ip[P] \leftarrow$ false
```

For a vector P, without performing specific calculations to obtain D_{QP}, the only information we can access are $D_{QC_{[P]}}$ and $D_{PC_{[P]}}$. The first distance is computed during the pre-search phase when identifying the *nprobes* nearest clustering centers to the query vector, while the second distance is pre-stored in the multi-granularity training module. Using these two distances, we can determine that D_{QP} lies between $D_{QC_{[P]}}$+$D_{PC_{[P]}}$ and $D_{QC_{[P]}}$-$D_{PC_{[P]}}$ by applying the triangle inequalities.

Example. Figure 4 shows two cases of pruning using the center distance-based pruning module. By estimating D_{QP}, we can avoid the computation for two types of distributed vectors and show the regions in $\mathbb{R}^2$: $D_{QC_{[P]}} - D_{PC_{[P]}} > TopK\_W$ (shown in Figure 4 (a)) and $D_{QC_{[P]}} + D_{PC_{[P]}} < TopK\_W$ (shown in Figure 4 (b)). We represent the possible locations of vector P at distance $D_{PC_{[P]}}$ from point $C_{[P]}$ with blue dashed lines. Regardless of the position of P, it does not enter the circular domain at a distance $TopK\_W$ from the point Q (shown as the black circle). Such a point P can be safely pruned without actually computing the distance from point Q. In Figure 4 (a), the region closer to D (the blue area) is pruned, while in Figure 4 (b), the farther region is pruned. Although Figures 4 (a) and (b) illustrate this in $\mathbb{R}^2$, we can generalize the same pruning principle to $\mathbb{R}^P$ for $P \geq 2$.

Limitation. Using only the center distance-based pruning module does not yield a good pruning ratio, as discussed in *distance calculation* in Section 3.2. As the dimension increases, the volume occupied by the center of the sphere in high-dimensional space becomes smaller, so Figure 4 (a) does not yield good pruning rates. Additionally, the number of vectors needed to achieve the same dataset density grows exponentially. Real datasets often do not provide such a large number of vectors, meaning that higher dimensions result in sparser clusters. Consequently, the probability that a query vector and its $TopK\_W$ neighborhood fall exactly into a cluster is greatly reduced.



(a) Pruning cluster center.

(b) Pruning cluster edge.

Fig. 4. Two different pruning situations.

Although Figure 4 (b) provides a larger pruning region compared to Figure 4 (a), the occurrence of the scenario depicted in Figure 4 (b) is much less frequent than that of the scenario in Figure 4 (a).

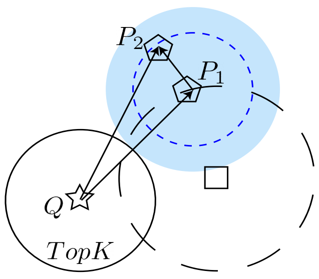
4.4 SubNN pruning module

From our previous analysis, we find that the triangle inequalities pruning brought by $D_{QC_{[P]}}$, $D_{PC_{[P]}}$, and $TopK\_W$ alone is not effective in checking all vectors based on distance. Since the scenario in Figure 4 (b) occurs much less frequently as dimensionality increases, we focus our further subNN pruning efforts on the scenario depicted in Figure 4 (a). To achieve a better pruning rate, it is necessary to prune vectors on the spherical shell, and a new pruning module needs to be introduced.

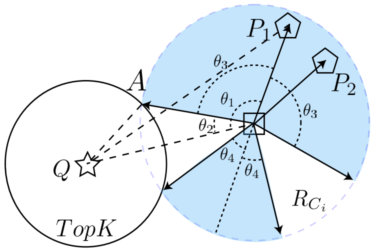
4.4.1 SubNN distance-based pruning module. We find that points near the spherical shell span a wide range of distances from Q due to their broad distribution. A straightforward approach is that if we encounter a vector P that is far from the query vector Q during computation, then a vector P_i that is closer to P will also not appear in the neighborhood of Q. To store each vector's neighboring vectors within the clusters, Tribase finds the respective top-K nearest neighbors of each vector inside each cluster after performing clustering. We call this process subNN search in the multi-granularity training module.

Example. We show an example of subNN distance-based pruning in Figure 5 (a), where P_1 is a vector that has already computed the distance to Q. If the distance between P_1 and P_2 has been preserved by subNN and if the distance between P_1 and P_2 satisfies $D_{QP_1} - D_{P_1P_2} > TopK\_W$, then we can also use triangle inequalities to prune P_2 without loss of precision. We denote the regions in the neighborhood of P_1 that can be pruned by the subNN distance-based pruning module in blue in Figure 5 (a). Note that P_1 can be any vector in the sphere. We find that, except for the part of the intersection between $B(Q, TopK\_W) - C_i$ (i.e., space in the cluster that will also be in $Top[K]$), everywhere else can be pruned. This solves the problem of not being able to prune the spherical shell in Figure 4 (a).

Drawback. While it is theoretically possible to prune any part of $B(Q, TopK\_W) - C_i$ using the subNN distance-based pruning module, as we analyzed in Section 3.2, as the dimension increases, the volume of a sphere in high-dimensional space is concentrated in the spherical shell. This results in the pruned spherical region occupying only a small fraction of the entire space, which is an inherent limitation of distance-based pruning. This issue arises because the radius of the pruned spheres would need to be as large as the clustering radius to be effective, which is not feasible. Therefore, the current pruning effectiveness shown in Figure 5 (a) still has significant room for improvement.



(a) SubNN distance pruning.



(b) SubNN angle pruning.

Fig. 5. Example of different kinds of subNN pruning.

4.4.2 SubNN angle-based pruning module. To overcome the limitations of distance-based pruning, we also consider angle-based pruning. As analyzed in Section 3.2, the angle between two random vectors in high-dimensional spaces has a high probability of being near 90°, which poses a challenge for angle-based pruning. This is because the direction of the vectors in a cluster is generally the same, so if one vector is perpendicular to the query vector, there is a high probability that the other vectors in the same cluster are also perpendicular to the query vector, reducing the effectiveness of pruning.

We note, however, that if we represent the vector P in the cluster as the residual of P and $C_{[P]}$, filtering the pruned vectors by the residuals will alleviate the problem of uneven distribution of angles within a single cluster. This is because vectors in the cluster can be viewed as emanating from the clustering center in all directions, and the angles between these vectors and the query vector will not be concentrated over a small interval.

Difficulties. There are two main difficulties in angle-based pruning. The first is that we cannot compute the angle between two vectors P and Q and prune them by calculating their inner product because the inner product computation and the L2 computation share the same complexity, $O(d)$, where d is the dimension of the vector. The second difficulty is that once the query vector is determined, Tribase must quickly identify the vectors within the cluster that meet the angular conditions to potentially qualify as Top-K candidates. This process is less straightforward than center distance-based pruning, as it requires more complex calculations.

Solution. To address the first difficulty, we choose to calculate the angle between two vectors using the cosine theorem. Since the cosine function itself does not satisfy the triangle inequalities, we need to convert the value of the cosine function into an angle to prune using the triangle inequalities. To solve the second difficulty, we introduce a benchmark vector by computing the angle between $B(Q, TopK\_W) - C_i$ and the benchmark vectors, and then classify the vectors accordingly based on the resulting angles. Specifically, we compute the angles between the Q and the random benchmark vector. Similar to center distance-based pruning, we then prune vectors within the cluster based on the results of these angle calculations. As shown in Figure 5 (b), we can also retain the nearest neighbors based on each vector's angle and accordingly prune the angle-based neighborhood of vectors within the cluster. In practice, we find that the former method is not very effective due to the influence of the curse of dimensionality, because most of the vectors are still close to 90° to the randomly specified benchmark vector, leading to poor classification results. However, the latter angle-based subNN obtains a good range of angular neighborhoods for each vector.

Example. We present an example of the subNN angle-based pruning module in Figure 5 (b). We take $\overrightarrow{QC_{[P_1]}}$ as the benchmark vector because $D_{QC_{[P_1]}}$ has been pre-computed in the assignment of the nearest *nprobe* clustering centers. Next, we calculate the range of angles, denoted as θ_2, formed by $B(Q, TopK\_W) - B(C_{[P_1]}, D_{C_{[P_1]}})$ and the benchmark vector, using the cosine theorem as follows:

$$\cos(\theta_2) = \frac{D^2_{QC_{[P_1]}} + D^2_{AC_{[P_1]}} - D^2_{QA}}{2 \cdot D_{QC_{[P_1]}} \cdot D_{AC_{[P_1]}}},$$

where different cases of θ_2 can be found in Appendix A.

After determining θ_2, we can prune the region where the angle with vector $\overrightarrow{QC_{[P_1]}}$ exceeds θ_2. Specifically, by computing D_{QP_1}, we can apply the cosine theorem to obtain the angle θ_1 between $\overrightarrow{QC_{[P_1]}}$ and $\overrightarrow{P_1C_{[P_1]}}$:

$$\cos(\theta_1) = \frac{D^2_{QC_{[P_1]}} + D^2_{P_1C_{[P_1]}} - D^2_{QP_1}}{2 \cdot D_{QC_{[P_1]}} \cdot D_{P_1C_{[P_1]}}}$$

For vector P_2, which lies within the angular neighborhood of P_1, two potential pruning regions may arise:

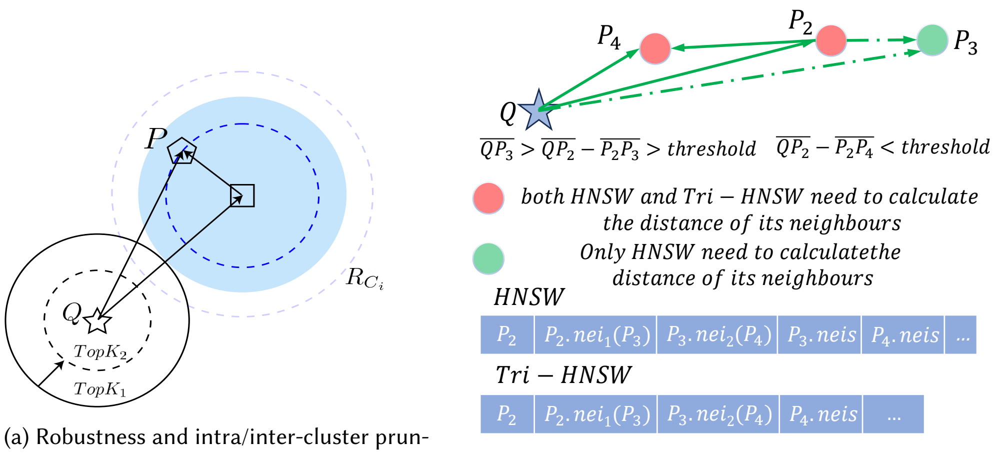
- If the angle between $\overrightarrow{P_1C_{[P_1]}}$ and $\overrightarrow{P_2C_{[P_1]}}$ satisfies the condition $\angle P_2C_{[P_1]}P_1 < \theta_1 - \theta_2$, then the angle between P_2 and $\overrightarrow{QC_{[P_1]}}$ becomes too small, preventing P_2 from being part of $B(Q, TopK\_W)$ (labled in region θ_3). Consequently, P_2 will fall within region θ_3, which can be pruned.
- If the angle between $\overrightarrow{P_1C_{[P_1]}}$ and $\overrightarrow{P_2C_{[P_1]}}$ satisfies the condition $\angle P_2C_{[P_1]}P_1 > \theta_1 + \theta_2$, then the angle between P_2 and $\overrightarrow{QC_{[P_1]}}$ becomes too lagre, also preventing P_2 from falling into $B(Q, TopK\_W)$ (labled in region θ_4). Consequently, P_2 will fall within region θ_4, which can be pruned.

These two pruning regions allow Tribase to perform pruning on distance and angle after calculating the distance between a point P_1 and a query point Q, significantly improving the pruning rate.

Limitation. Although the subNN angle-based pruning module can provide a dimension-independent pruning effect guarantee, it introduces a few more floating-point multiplications compared to the center distance-based pruning module and the subNN distance-based pruning module, since we need to use the cosine theorem to get the cosine value through the Euclidean distance. Moreover, existing vector databases usually use SIMD instructions to handle multiple dimensions simultaneously, which reduces the overhead of the original distance computation. Therefore, the overhead of introducing the subNN angle-based pruning module is not as negligible as that of the center distance-based pruning module and the subNN distance-based pruning module. We recommend using the subNN angle-based pruning module cautiously for low-dimensional data and refer to our experiments in Section 6.3.1 for details.

4.5 Analysis of pruning strategies

Design of pruning intensity. Since the equality sign requirement of the triangle inequalities is very strict in high-dimensional spaces, we can appropriately tighten the pruning ratio in the pruning condition shown in Figure 2 (b) to achieve stronger pruning. However, doing so will inevitably filter out some high-quality vectors that originally existed in the results. This approach has two advantages: First, the original pruning is too conservative, and by tightening the pruning ratio, a larger amount of pruning can be achieved. Second, the distance between the vectors incorrectly pruned by this method and the query vectors are usually near the bound of topK, having a minimal


P
R_{C_i}
Q
$TopK_2$
$TopK_1$
P_4
P_2
P_3
Q
$\overline{QP_3} > \overline{QP_2} - \overline{P_2P_3} > threshold$
$\overline{QP_2} - \overline{P_2P_4} < threshold$
both HNSW and Tri − HNSW need to calculate the distance of its neighbours
Only HNSW need to calculatethe distance of its neighbours
HNSW
P_2 | $P_2.nei_1(P_3)$ | $P_3.nei_2(P_4)$ | $P_3.neis$ | $P_4.neis$ | ...
Tri − HNSW
P_2 | $P_2.nei_1(P_3)$ | $P_3.nei_2(P_4)$ | $P_4.neis$ | ...

(a) Robustness and intra/inter-cluster pruning example.

(b) Graph pruning example.

Fig. 6. Pruning strategies in various scenarios.

overall impact on the quality of the results. In Section 6.4.1, we show the improvement in the pruning ratio and the impact on the results using different pruning intensities.

Robustness and correctness. We first prove the correctness of our pruning method, which is based on the fact that $TopK\_W$ gets progressively better during the computation. As shown in Figure 6 (a), $TopK\_W$ decreases from $TopK\_W_1$ to $TopK\_W_2$ during the pruning process. We find that in the distance-based pruning modules, regardless of whether it is the center distance-based pruning module or the subNN distance-based pruning module, our pruning always follows the triangle inequality $D_{PC_{[P]}} + TopK\_W < D_{QC_{[P]}}$ or $D_{PP.neighbor} + TopK\_W < D_{PQ}$. The correctness of both pruning schemes is not affected by the fact that $TopK\_W$ gets smaller. In the subNN angle-based pruning module, we prune the vectors in the region whose angle to $\overrightarrow{P_1C_{[P_1]}}$ is $[0, \theta_1 - \theta_2]$ and $[\theta_1 + \theta_2, \pi/2]$. As analyzed in Appendix A, the partial derivatives of $TopK\_W < 0$, so when $TopK\_W$ becomes smaller, the cosine value of the angle θ_2 will become larger, and angle θ_2 will become smaller, ensuring that the triangle inequalities is satisfied.

Intra-cluster and inter-cluster pruning. Although Tribase's pruning granularity is based on individual vectors, it does not mean it cannot prune at the cluster granularity level, as shown in Figure 6 (a). As the search proceeds, when there is no intersection between the newly added clusters and the $TopK\_W_2$ neighborhood of the query point Q, each vector in the clusters where $C_{[P]}$ is located can be pruned by the triangular inequalities $D_{PC_{[P]}} + TopK\_W < D_{QC_{[P]}}$ without distance computation, thus completing the pruning at the cluster granularity level.

4.6 Pruning in graph-based index

Graph-based index has gained significant attention due to its efficient search performance [8, 19, 30, 45]. Hierarchical Navigable Small World (HNSW) [45], a popular algorithm for ANNS, based on small-world graphs, is widely used for indexing high-dimensional vector spaces. It constructs a multi-layered graph and leverages small-world network properties to achieve fast convergence when finding nearest neighbors in high-dimensional spaces.

We observe opportunities for optimization in HNSW by applying triangle inequalities to accelerate graph-based searches. As shown in Figure 6 (b), the current HNSW implementation adds all neighbors of a vector P_2 to the search queue while calculating the distance between Q and P_2. This can lead to inefficiencies when low-quality vectors like P_3 and its neighbors are added to the search queue, reducing search efficiency.

To address this, we introduce a pruning threshold, applying triangle inequality pruning to P_2's neighbors whose distances exceed this threshold. Specifically, if $D_{QP_2} - D_{P_2P_3} > threshold$, we can prevent P_3 and its neighbors from being prematurely added to the queue, accelerating the ANNS process in graph-based indexing.

We extend HNSW's original search logic with this triangle inequality pruning approach, resulting in a new graph-based search method called TriHNSW. In Section 6.2.3, we compare the performance of TriHNSW with HNSW.

5 Implementation

Tribase is implemented in C++20 and comprises about 6,000 lines of code. It utilizes modern C++ features, such as compile-time computation and zero-cost abstractions, provided by the new standard to enhance execution efficiency and code readability. The implementation of Tribase consists of two phases. In the training phase, Tribase uses the same clustering algorithm and parameters as the Faiss [17] library to verify the validity of pruning. The difference is that Tribase simultaneously performs secondary clustering within the clusters at the end of clustering to compute and retain the relevant subNN fields. In the query phase, Tribase supports commonly used high-performance mathematical computing libraries, including Eigen and Intel MKL, along with handwritten instruction sets, to perform rapid geometric calculations such as the cosine theorem and the judgment of different pruning boundaries. Note that since the pruning in Tribase itself does not require partial distance calculations between vectors, it is well compatible with the SIMD instruction set. For fast computation, we support the SIMD instruction set in Tribase and use the Intel MKL library to optimize parallel operations. We next discuss APIs and indexing parameters.

Parameters. Since Tribase provides a variety of unique parameters, to make it easier for users to familiarize themselves with and use them, we hide the internal pruning details and present the same interface as a typical ANNS system [8, 17, 21, 76]. We also offer suggestions on parameter selection. For indexing-related parameters, we have two key findings: First, while theoretically, building the subNN index does not significantly increase the indexing time, the quality and time overhead of the SubNN index can be further controlled through the *sub-nprobe-ratio* parameter we provide. In Section 6.4.3, we conducted experiments to analyze this aspect. Second, setting *subK* greater than 10 does not significantly improve subsequent pruning. Users can modify the size of *subK* and subNN search intensity *sub-nprobe-ratio* to achieve different training speeds and indexing qualities according to their actual needs. We show the effect of subNN indexes of different quality on subsequent pruning in Section 6.4.2. For search-related parameters, we suggest that if users want to minimize indexing overhead, enabling only *enable Triangle* can provide a better trade-off between training time and pruning effect. If users do not mind the training time, they can enable all three parameters—*enable Triangle*, *enable SubNNL2*, and *enable SubNNIP*—to achieve the highest pruning rate. The choice of pruning modules can be referenced from the experiments in Sections 6.2 and 6.3.1. For the pruning ratio parameter, if the requirement for the final result is not particularly high (e.g., more than 99.9%), we suggest setting the pruning rate at 0.8 for a better pruning effect. Otherwise, setting the pruning ratio to 1.0 guarantees completely lossless pruning. The choice of pruning ratio can be referenced from the experiments in Section 6.4.1.

6 Evaluation

6.1 Experiment setup

Methodology. We primarily compare Tribase against Faiss, the state-of-the-art ANNS library [17] with SIMD enabled. We compared the performance of Faiss with several configurations of Tribase: Tribase with only the center distance-based pruning module enabled (abbreviated as Tribase-Triangle), Tribase with only the SubNN distance-based pruning module enabled (abbreviated as

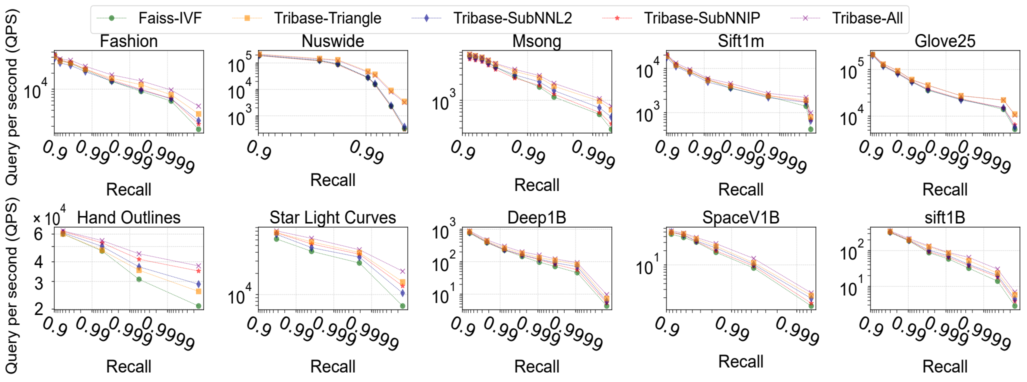


Fig. 7. Time-accuracy trade-off.

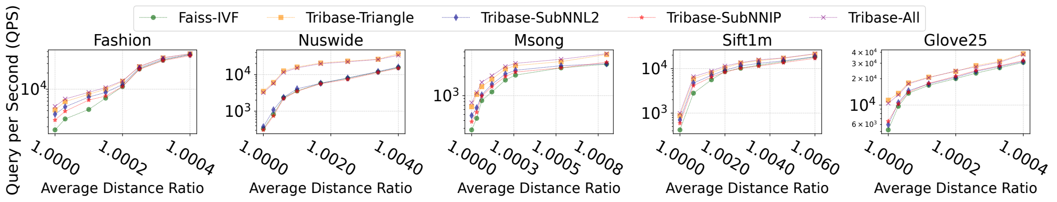


Fig. 8. Time-accuracy trade-off (Average distance ratio).

Tribase-SUBNNL2), Tribase with only the SubNN angle-based pruning module enabled (abbreviated as Tribase-SUBNNIP), and Tribase with all pruning modules enabled (abbreviated as Tribase-All). To fully assess the advantages of our work, Tribase uses the same clustering method and number of clusters as Faiss. After performing the same clustering operations, we measure the time overhead of obtaining the same results using Faiss and lossless versions of Tribase, as well as the time overhead of Faiss and lossy versions of Tribase with the same results. Some works [51, 67] also utilize the triangle inequality to accelerate ANNS, such as the pruning in VAQ [51], which operates on clusters of encoded data. We compare the pruning strategy in Tribase with that in VAQ. Another work, ADSampling [21], accelerates ANNS by partially calculating the distance between two vectors, allowing for pruning during individual distance calculations. In this paper, we compare the performance of Faiss, Tribase, and ADSampling with and without SIMD acceleration. Since high-precision ANNS is the big trend, we mainly focus on the performance difference between Faiss and Tribase in high-precision scenarios.

Datasets. We use ten open-source datasets of different sizes and dimensions, as shown in Table 2. These datasets are created by embedding various types of data (image, sound, and text) with different dimensions, providing both data and query vectors. They have been evaluated by many ANN systems [8, 76] and algorithms [19, 21, 45].

Platform. We conduct a standard performance evaluation of Tribase on a server equipped with an Intel(R) Xeon(R) Gold 5318Y CPU, featuring 24 cores and 96 threads. The server has 128GB of global memory, and the operating system used is CentOS Stream 8.

6.2 Overall performance

6.2.1 Time-accuracy trade-off. In this part, we analyze the trade-off between query per second (QPS) and recall when comparing Faiss with Tribase. First, we analyze the overall search quality and

Table 2. Dataset statistics.

Dataset	Size	D	Query Size	Data Type
Fashion	60,000	784	10,000	Image
Nuswide	268,643	500	200	Image
Msong	992,272	420	1,000	Audio
Sift1M	1,000,000	128	10,000	Image
Glove25	1,183,514	25	10,000	Text
Hand Outlines	1000	2709	370	Time Series
Star Light Curves	8236	1024	1,000	Time Series
Deep1B	1,000,000,000	96	10,000	Image
SpaceV1B	1,000,000,000	100	10,000	Text
Sift1B	1,000,000,000	128	10,000	Image

throughput on different datasets between Tribase and Faiss, since both Tribase and Faiss allow users to make a trade-off between search quality and throughput. The experimental results are shown in Figure 7. We have the following four findings: First, for all datasets, the pruning modules using center distance-based pruning, subNN distance-based pruning, and subNN angle-based pruning alone gain clear speedups, with center distance-based pruning alone having the best results. Second, for the same dataset, the speedup ratio of Tribase compared to Faiss rises with accuracy, reaching a speedup of 3.11× on average for *recall*=1. This is because as the search targets higher accuracy, the newly added vectors and their clustering centers inevitably increase their distance from the query vector, making the pruning of these vectors more effective. Third, since the pruning modules of Tribase are independent of each other, we can combine some of them to achieve higher speedups. For example, on the *msong* dataset, using all the pruning modules simultaneously achieves a achieve a performance improvement of 19% compared to using center distance-based pruning alone. Fourth, as the dimensionality of the dataset increases, the pruning effect of Tribase-SubNNL2 and Tribase-SubNNIP becomes more pronounced. This is because higher dimensionality results in increased complexity for vector distance calculations, which in turn reduces the relative impact of the additional fixed computations introduced by Tribase.

6.2.2 Time-distance trade-off. In this part, we analyze the time-distance trade-off. As mentioned in Section 4.5, Tribase can achieve significant speedup ratios at the expense of a small amount of accuracy by adjusting the pruning strength of the pruning modules. We further analyze the relationship between the speedup ratios and the quality of the results by setting different pruning intensities for each dataset. The experimental results are shown in Figure 8. We have the following three findings: First, for different datasets, both Tribase and Faiss can achieve more than 10× speedup by sacrificing a small amount of precision. However, compared to Faiss, Tribase not only controls the precision by modifying the number of clusters in the query but also adjusts the pruning strength, which allows for controllable sub-optimal results. Second, adjusting the pruning strength of Tribase and changing the number of search clusters can lead to higher query speeds than simply changing the number of search clusters for the same query results in Faiss. Third, Tribase provides a higher speedup ratio compared to Faiss when offering higher search quality. This is because achieving higher search quality requires querying more low-quality clusters, whereas Tribase can efficiently prune vectors in low-quality clusters. Due to space limits, we present the experimental results for only the first five datasets. The others can be find in Appendix G.

6.2.3 Comparation with graph-based index. In Section 4.6, we introduced the application of triangle inequality in graph-based indexing and applied this concept to the mainstream HNSW search engine, hnswlib [44], resulting in TriHNSW. Next, we compare the search performance of TriHNSW and HNSW, and the experimental results are shown in Figure 9.

We have three findings. First, while both TriHNSW and HNSW exhibit performance fluctuations across different datasets, TriHNSW shows smaller fluctuations due to its proactive elimination of low-quality vectors and their neighbors. Second, overall, TriHNSW achieves a clear speedup over HNSW, with a 25% performance improvement at recall=95. Third, the performance and stability of graph-based algorithms vary across different datasets, which is consistent with previous experiments [19, 44, 45].

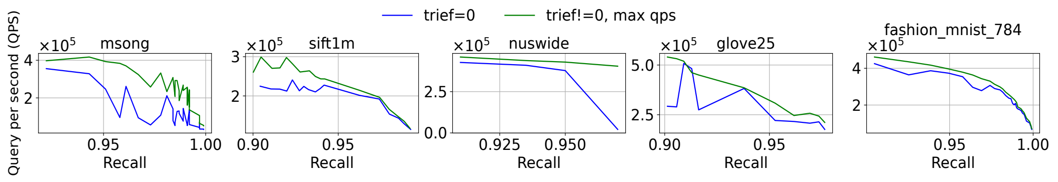


Fig. 9. Performance between HNSW and TriHNSW.

6.3 Performance ablation study

6.3.1 Pruning ratio breakdown. We analyze the effect of each pruning module and the time overhead it brings in Tribase in this section. We measure the various pruning modules of Tribase when they work individually and together on each dataset losslessly with *recal*=1 (noting that lossy pruning results in a larger pruning rate), as shown in Figure 10. For each pruning module on each dataset, we use red bars to denote the percent of lossless pruning vectors divided by the total number for that module, and blue bars to denote the additional time overhead associated with that pruning module. For example, for the pruning method of *Tribase-Triangle* in the *msong* dataset, which increases the time overhead by 3.24% while achieving a 63.31% pruning ratio, the speedup ratio is $\frac{1}{1-63.31\%+3.24\%} = 2.50$.

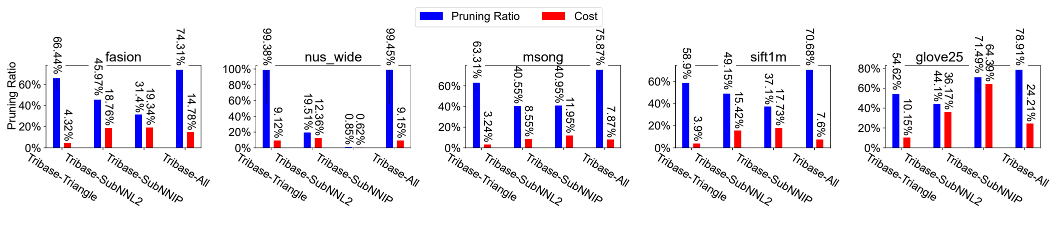


Fig. 10. Performance benefit breakdown on pruning modules.

We have the following findings. First, when obtaining the same or similar pruning results, the additional overhead introduced by the center distance-based pruning module is less than that of the subNN distance-based pruning and subNN angle-based pruning modules. This is because, although Tribase precomputes and stores some of the fields required for pruning in the training module, the subNN distance-based and subNN angle-based pruning modules still introduce several multiplicative computations compared to the center distance-based pruning module. Second, since different datasets are derived from different embedding methods and have different distributions, the most effective pruning methods vary across datasets. This further justifies our introduction of the subNN distance-based and subNN angle-based pruning modules based on the center distance-based pruning module. Third, the combined pruning ratio of introducing center distance-based pruning, subNN distance-based pruning, and subNN angle-based pruning modules simultaneously is smaller than the sum of the pruning ratios obtained by introducing these three modules separately. This is because the three pruning modules will prune some of the same vectors repeatedly.

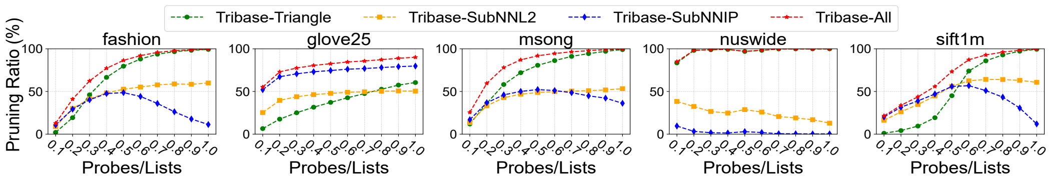


Fig. 11. Impact of pruning effectiveness of various modules on the search phase.

6.3.2 *Relationship between pruning ratio and search probes.* Due to the different strategies of various pruning modules, the pruning effect of each module changes as the search proceeds. Next, we analyze the pruning rate of each module at different stages of the search, so users can determine whether to use pruning at a particular stage. We show the corresponding pruning ratios for each module at different stages of the search in Figure 11. The horizontal axis indicates the search stage, while the vertical axis indicates the effect of pruning. For example, (0.2, 80) means that the pruning rate is 80% when searching from 0.1×lists to 0.2×lists clusters.

We have the following three findings. First, as the search proceeds, the pruning effect of the center distance-based pruning module improves more significantly compared to the subNN distance-based and subNN angle-based pruning modules. This is because the pruning effect of the center distance-based pruning module is positively correlated with the distance between the clustering center and the query point. As the search proceeds, the newly searched clusters become increasingly distant from the query point. Second, when enabling all three modules simultaneously, most of the data reach a 50% pruning rate before completing 30% of the search and an 80% pruning rate when 50% of the search is complete. This implies that if all the pruning modules of Tribase are enabled, unnecessary vector distance calculations can be effectively reduced in high-precision searches, greatly saving query time, as shown in the experimental results of Section 6.2. Third, although the pruning effect of the subNN distance-based and subNN angle-based pruning modules does not increase as the search proceeds, they provide a better pruning effect in the early stages of the search compared to the center distance-based pruning module.

6.3.3 *Impact of dataset size and dimensionality on pruning effectiveness.* We examine the impact of dimensionality and dataset size on Tribase's pruning effectiveness. To eliminate the influence of different data distributions on the pruning strategy, we create datasets with uniform distribution but differing dimensionalities, all maintaining the same density, to validate the impact of data dimensionality on pruning effectiveness. Additionally, we build two datasets with different densities for dimensions 4 and 8 to test the impact of dataset size on pruning effectiveness. The datasets and their corresponding pruning results are shown in Table 3, where "i d j s" represents an i-dimensional dataset containing j vectors.

Table 3. Impact of dimensionality and size on pruning effectiveness.

Size/Dimention	**8d256s**	**8d65536s**	**16d65536s**	**20d1048576s**
pruning_ratio (%)	61.99	96.39	68.67	75.16

We have the following two observations. First, as data dimensionality increases while maintaining the same density, the pruning effect of Tribase increases. For example, in dataset 8d256s, Tribase achieves a pruning ratio of 61.99%, whereas in dataset 20d1048546s, with the same density, the pruning ratio rises to 75.16%. This is because Tribase prunes regions outside of the overlapping TopK neighborhoods, and due to the curse of dimensionality, the overlapping region occupies a smaller proportion of the total cluster area. Second, as the density of data with the same dimensionality increases, the pruning effect of Tribase also increases. For instance, in dataset 8d256s, Tribase achieves a pruning ratio of 61.99%, while in dataset 8d65536s with the same dimensionality, the pruning ratio increases to 96.39%. This is because denser data distributions within a cluster allow Tribase to construct more effective subNN indices. These results demonstrate Tribase's potential for handling large, high-dimensional datasets.

6.4 Results of parameter study

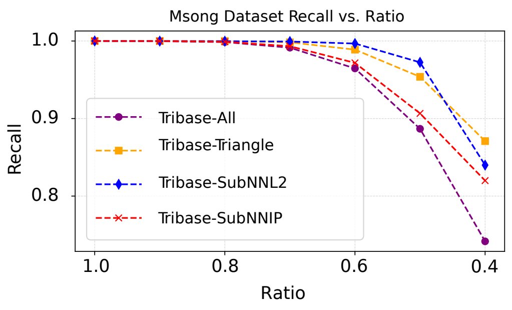
6.4.1 Trade-off between pruning intensity and accuracy. Since Tribase can trade-off between the quality of the search results and the speed of the search by varying the intensity of the pruning in addition to the number of clusters searched, we discuss the effect of using different pruning intensities on all pruning modules in Tribase on the *msong* dataset, as the three pruning modules in this dataset have similar pruning ratios. The experimental results are shown in Figure 12 (a). We have the following three findings:

First, using a lower intensity of lossy pruning can be done with little or no impact on the quality of the results, but as the intensity of lossy pruning rises, the quality of the final results will show a significant decrease. Second, if the same intensity of pruning is applied to all three pruning modules, the quality of the final results will be worse than applying the same intensity to any single pruning module. Third, the center distance-based pruning module reduces accuracy less than the subNN distance-based pruning module and subNN angle-based pruning module when using low-intensity lossy pruning. However, as the intensity of pruning increases, the results of the center distance-based pruning module decline more than those of the subNN distance-based pruning and subNN angle-based pruning modules.

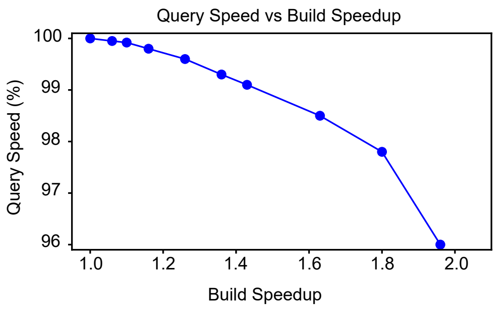
Based on these findings, we can adopt different intensity pruning strategies at different stages of the query. For high-quality clusters searched at the beginning of the search, cautious lossy or lossless pruning should be used. For low-quality clusters, aggressive pruning strategies can be employed based on the need for the results. The intensity of the pruning should guide the selection of the pruning method that minimizes the impact on the results.

6.4.2 Trade-off between build time and query time. As discussed in Section 4.2, when the subNN pruning module is enabled, the subNN search for each vector in its cluster can provide either a fine or coarse subNN index. A fine subNN index requires longer training time but offers better indexing and reduces the time spent on ANN search. To balance different pre-training times and query times, Tribase supports adjustable subNN index construction parameters in the multi-granularity training module. The results of subNN indexing with different parameters in terms of pre-training time and query effectiveness are shown in Figure 12 (b).

We find that coarse subNN indexes have little impact on the query compared to fine indexes, and even accelerating the subNN process by 1.8× only impacts the query speed by less than 2%. This is because, on one hand, the pruning acceleration provided by the subNN pruning modules is limited, and on the other hand, since the build of subNN indexes is also a TopK query, executing a suboptimal TopK query has a small impact on the results compared to the optimal subNN indexes. However, we still recommend that users build the optimal subNN index without prioritizing training time to achieve the best query performance.



(a) Relationship between pruning ratio and search probes.



(b) Results of subNN indexes with different parameters.

Fig. 12. Impact of pruning effectiveness and indexing parameters on search performance.

6.4.3 Indexing time usage with different parameters. As discussed in Section 5, we can control the search scope when building the subNN index by adjusting the *sub-nprobe-ratio* parameter. This allows for either a full search or ANNS within the cluster.

We compare the time required for Faiss, Tribase-Triangle, and two SubNN indexing scenarios: one with *sub-nprobe-ratio*=1, which performs a full subNN neighbor search, and the other with *sub-nprobe-ratio*=0.5, which performs a high-accuracy approximate subNN neighbor search. Our results are shown in Table 4. Since the update process mirrors the steps of index construction, they have the same time overhead.

Table 4. Indexing time comparison.

Dataset	**Fashion**	**Nuswide**	**Msong**	**Sift**	**Glove**
Faiss	2.06	4.66	13.59	7.63	6.90
Triangle	2.08	4.67	13.65	7.67	6.93
SubNNL2 ratio 0.5	3.14	14.38	37.23	25.19	23.46
SubNNIP ratio 0.5	3.30	19.15	55.98	22.98	21.23
All ratio 0.5	5.34	28.27	81.84	45.90	43.72
SubNNL2 ratio 1	3.61	17.07	46.62	30.02	28.79
SubNNIP ratio 1	4.04	24.49	70.72	27.13	25.38
All ratio 1	7.09	35.57	101.43	53.55	52.46
HNSWlib & TriHNSW	5.09	36.78	123.49	63.55	59.66

We have four key observations. First, enabling only Tribase-Triangle does not result in a significant difference in indexing time compared to Faiss, as the fields required by Tribase-Triangle can be naturally derived from Faiss's clustering. Second, different *sub-nprobe-ratio* parameters lead to varying indexing times; on average, using *sub-nprobe-ratio*=1 takes 1.93 times longer than using *sub-nprobe-ratio*=0.5, which corresponds to the search overhead of their respective subNN indexes. Third, under the same parameters, the indexing time for Tribase-All can approximately be seen as the sum of the times for Tribase-subNNL2 and Tribase-subNNIP, as Tribase-All requires both angle and distance-based searches. Fourth, both HNSW and TriHNSW have the same time overhead, as TriHNSW does not require additional precomputations on top of the HNSW index. Furthermore, graph-based indexes like HNSW have certain advantages over cluster-based indexes like Faiss on smaller datasets. However, as the dataset size increases, the index construction time for graph-based indexes grows faster compared to that of cluster-based indexes.

6.5 Memory usage

6.5.1 Index memory usage. As discussed in Section 4.2, Tribase requires additional space to store the subNN fields in order to perform SubNN distance-based and SubNN angle-based pruning. We conduct experiments to evaluate the impact of SubNN index parameters on index size, with the results present in Table 5.

We have four key findings. First, the space overhead introduced by the SubNN index is minimal compared to the size of the vectors themselves. For example, in the msong dataset, when the *SubK* parameter is set to 7, Tribase-SubNNL2 index requires only 5.1% additional space beyond the original vectors. Second, Tribase-SubNNIP index consumes more space than the Tribase-SubNNL2 index. For instance, in the msong dataset, with the *SubK* parameter set to 7, Tribase-SubNNIP index requires 4.6% more space than the Tribase-SubNNL2 index. Third, the space occupied by the SubNN index of Tribase is related only to the number of vectors and is independent of the dimensionality. For example, in the sift1m and msong datasets, since the number of vectors in these two datasets are similar, the index overhead with the same SubNN parameters is also comparable. Specifically, building Tribase-SubNNIP index of both datasets requires approximately 160MB more space than Faiss. However, due to the higher dimensionality of the msong dataset, this overhead is relatively smaller compared to the storage cost of the index itself. This demonstrates the potential of Tribase in high-dimensional datasets for deployment in space-constrained scenarios. Fourth, the space overhead of using the HNSW index is roughly between those of SubNNL2 and SubNNIP. The space overhead for TriHNSW is approximately twice that of HNSW. This is because TriHNSW requires pre-storing a distance on each edge in HNSW, effectively doubling the storage cost for each edge.

Table 5. Index memory comparison.

Dataset	Fashion	Nuswide	Msong	Sift	Glove
Faiss	181MB	517MB	1.57GB	501MB	127MB
Triangle	182MB	519MB	1.57GB	508MB	136MB
SubNNL2	192MB	538MB	1.65GB	581MB	222MB
SubNNIP	202MB	562MB	1.73GB	669MB	326MB
All	213MB	584MB	1.81GB	749MB	421MB
HNSW	188MB	550MB	1.69GB	630MB	281MB
TriHNSW	375MB	1.07GB	2.40GB	1.22GB	551MB

6.5.2 Peak memory usage. Since Tribase saves more distance and angle fields for subsequent pruning when constructing the index compared to Faiss, the peak memory required during the entire process of indexing and querying may exceed that of Faiss. We perform end-to-end comparisons of the whole process from constructing the index to completing the query using the different pruning modules of Tribase and Faiss, respectively. The experimental results are shown in Table 6.

We have the following three findings. First, overall, Faiss and Tribase-Triangle possess lower peak memory usage. Tribase-SubNNL2 is the next highest, with 1.29 times the peak memory of Faiss on average, and Tribase-SubNNIP and Tribase-All have the highest peak memory, with 1.76 times the peak memory of Faiss on average. Second, we find that the peak memory of all methods is proportional to the size of the dataset itself. Relatively speaking, Tribase adds a smaller percentage of peak memory on high-dimensional vectors than on low-dimensional vectors. This is because the field added by Tribase possesses a fixed size, and the higher the dimension of the data itself, the lower the share of this part of the data, demonstrating the potential of Tribase in processing high-dimensional data. Finally, from a memory footprint perspective, we do not recommend using only the subNN angle-based pruning module, as enabling the remaining two pruning modules simultaneously does not result in significant additional memory overhead. It is worth noting that all modules in Tribase generate some intermediate results, which introduce an additional overhead on top of the original indexing cost. However, since each module also contributes a certain pruning rate, it reduces some of the intermediate results inherent in ANNS. For example, in the Fashion dataset, the peak memory usage of Tribase-All is lower than that of Tribase-SubNNIP. This is because the Tribase-Triangle and Tribase-SubNNL2 modules in Tribase-All eliminate a significant amount of unnecessary computation, outweighing the additional overhead introduced by these two modules.

Table 6. Peak memory comparison.

Dataset	**Faiss**	**Triangle**	**SubNNL2**	**SubNNIP**	**All**
Fashion	469MB	458MB	556MB	717MB	716MB
Nuswide	1.06GB	1.06GB	1.36GB	1.73GB	1.75GB
Msong	3.32GB	3.33GB	3.89GB	4.66GB	4.73GB
Sift	1.17GB	1.19GB	1.37GB	1.66GB	1.79GB
Glove	396MB	403MB	648MB	865MB	1.07GB

6.6 Additional discussion

6.6.1 Integration with Faiss. To verify the scalability of the pruning logic in Tribase, we integrate the pruning modules of Tribase into Faiss. We modify the training and querying process of the original Faiss indexes of type IVFFlat, and, as in Tribase, we pre-save the fields required by the center distance-based pruning module, subNN distance-based pruning module, and subNN angle-based pruning module during training. We obtain similar results as in Section 6.2. This is because, in cluster-based indexing, 99.21% of the time in an ANNS is spent on calculating the distance between two vectors. Due to the superior optimization of Tribase and Faiss in this aspect, direct ANNS searches on Tribase and Faiss take almost the same time. Therefore, the time reduced by the pruning and the overhead incurred is similar to the results in Tribase.

It is important to note that we only add pruning to nodes of type Faiss-IVFFlat, not Faiss-IVFPQ. On the one hand, since IVFPQ reduces the dimension of the original vectors and involves lossy compression, the triangle inequalities that are satisfied after compression may not be satisfied before compression. On the other hand, since the dimensions of the vectors are reduced, less computation is required to compute the distance between two vectors. However, the additional computation required for pruning itself is fixed and more suitable for dealing with high-dimensional vectors, as verified in our experiments in Section 6.2. Thus, it is not appropriate to apply the pruning of Tribase on Faiss-IVFPQ index.

6.6.2 Comparison with ADSampling. ADSampling is the state-of-the-art in transformation-based work.

We compare the performance of Tribase, Faiss, and ADSampling with and without SIMD acceleration in Figure 13. We observed that when SIMD is not utilized, both Tribase and ADSampling outperform Faiss. They achieve acceleration ratios exceeding 2×, indicating the benefit of their respective pruning strategies. Tribase's performance falls between ADSampling-IVF++ and ADSampling-IVF+. This can be attributed to Tribase's strong pruning efficiency, while ADSampling-IVF++ is specifically optimized with a cache-friendly data structure. Second, when SIMD is enabled, both Tribase and Faiss achieve a speed up exceeding 2×, while ADSampling shows an average acceleration of 1.3×. This is because Tribase and Faiss perform full vector distance searches, whereas ADSampling

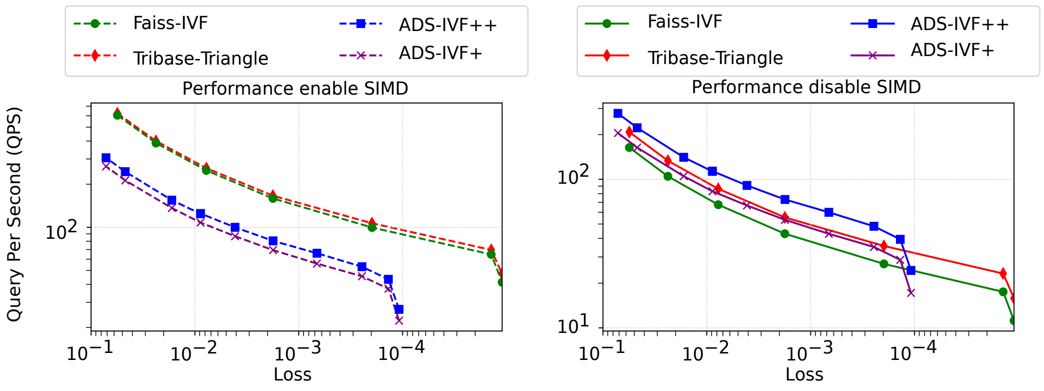


Fig. 13. Performance comparison of Faiss, Tribase, and ADSampling.

interrupts the process to check pruning conditions during the search, resulting in a slightly lower overall speedup.

Additionally, we compare the indexing time of Tribase, FAISS, and ADSampling. ADSampling requires nearly 10× more indexing time than Tribase-Triangle and FAISS due to the need to rotate vectors using a set of standard orthogonal bases.

6.6.3 Compare with other works using triangle inequalities. Several works have attempted to accelerate ANNS using triangle inequalities [51, 67]. Among them, VAQ [51] leverages distance-based triangle inequality pruning for cluster-based indexes. We compare the pruning and speedup ratio between Tribase-All and VAQ. We find that Tribase achieves a higher pruning ratio than VAQ due to its ability to perform multidimensional pruning based on both angular and distance metrics and further subdividing the clusters. During the search with 0.5 probe/lists, Tribase achieves 41% more pruning rate than VAQ on average, leading to a 36% increase in speedup. Additionally, Tribase offers controllable pruning intensity, allowing further acceleration of ANNS at the cost of slightly reducing accuracy. When achieving the same quality as VAQ, Tribase achieves over 50% speedup.

7 Conclusion

This paper introduces Tribase, an efficient method for lossless pruning of vector topK queries using triangle inequalities. By analyzing high-dimensional triangle inequalities, the curse of dimensionality, and vector index structures, we design three different pruning modules to capture low-quality vectors located at different positions within the clusters. These modules can provide more than 10× speedup and 99.4% pruning ratios during computation, with no more than 1.76 times the original memory overhead on average. Additionally, Tribase demonstrates strong potential for vector pruning in high-dimensional spaces due to its fixed index size and pruning overhead. Experimental results validate the promising utility of Tribase.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62322213 and 62172419), Beijing Nova Program (No. 20230484397 and 20220484137), the State Key Laboratory of Computer Architecture (ICT, CAS) under Grant CLQ202414 and CLO202414, CCF - ApsaraDB Research Fund (CCF-Aliyun2024003), and Public Computing Cloud, Renmin University of China. Feng Zhang is the corresponding author of this paper.

References

[1] 2024. UCI Machine Learning Repository. https://archive.ics.uci.edu/.

[2] 2024. YouTube. https://www.youtube.com/. https://www.youtube.com

[3] Artem Babenko, Victor Lempitsky, B.Hari Babu, N.Subhash Chandra, and T.V. Gopal. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.

[4] Francis R. Bach. 2017. Breaking the Curse of Dimensionality with Convex Neural Networks. *J. Mach. Learn. Res.* 18 (2017), 19:1–19:53. http://jmlr.org/papers/v18/14-546.html

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

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

[7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL] https://arxiv.org/abs/2005.14165

[8] 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. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J.Wortman Vaughan (Eds.). Vol. 34. Curran Associates, Inc, 5199–5212.

[9] Zheng Chen, Feng Zhang, Yang Chen, Xiaokun Fang, Guanyu Feng, Xiaowei Zhu, Wenguang Chen, and Xiaoyong Du. 2024. Enabling Window-Based Monotonic Graph Analytics with Reusable Transitional Results for Pattern-Consistent Queries. *Proc. VLDB Endow.* 17, 11 (Aug. 2024), 3003–3016. doi:10.14778/3681954.3681979

[10] Kenneth L. Clarkson. [n. d.]. An algorithm for approximate closest-point queries. In *Proceedings ofthe tenth annual symposium on Computational geometry.* 160–164,1994.

[11] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Va-hab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG '04.* 253–262,.

[12] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80.

[13] B.N. Delaunay. 1934. Sur la sphère vide. *Bull. Acad. Sci. URSS* 6 (1934), 793–800.

[14] Yufei Ding, Lin Ning, Hui Guan, and Xipeng Shen. 2017. Generalizations of the theory and deployment of triangular inequality for compiler-based strength reduction. *SIGPLAN Not.* 52, 6 (jun 2017), 33–48. doi:10.1145/3140587.3062377

[15] Yufei Ding, Lin Ning, Hui Guang, Xipeng Shen, and Madan Musuvathi. 2018. TOP : A Compiler-Based Framework for Optimizing Machine Learning Algorithms through Generalized Triangle Inequality. https://api.semanticscholar.org/CorpusID:6500348

[16] Wei Dong, Moses Charikar, 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, WWW2011.* Association for Computing Machinery, Hyderabad, India, 577–586. doi:10.1145/1963405.1963487

[17] Facebook. 2020. Faiss. https://github.com/facebookresearch/faiss.

[18] Cong Fu and Deng Cai. 2016. EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph.

[19] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.* 12, 5 (jan 2019), 461–474. doi:10.14778/3303753.3303754

[20] K.Ruben Gabriel and Robert R. Sokal. 1969. A new statistical approach to geographic variation analysis. *Systematic zoology* 18, 3 (1969), 259–278.

[21] Jianyang Gao and Cheng Long. 2023. High-Dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations. *Proc. ACM Manag. Data* 1, 2, Article 137 (jun 2023), 27 pages. doi:10.1145/3589282

[22] Soumya Suvra Ghosal, Yiyou Sun, and Yixuan Li. 2024. How to Overcome Curse-of-Dimensionality for Out-of-Distribution Detection?. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (Eds.). AAAI Press, 19849–19857. doi:10.1609/AAAI.V38I18.29960

[23] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 518–529.

[24] Wayne D. Gray and Deborah A. Boehm-Davis. 2000. Milliseconds matter: An introduction to microstrategies and to their use in describing and predicting interactive behavior. *Journal of experimental psychology: applied* 6, 4 (2000).

[25] Jiawei Guan, Feng Zhang, Siqi Ma, Kuangyu Chen, Yihua Hu, Yuxing Chen, Anqun Pan, and Xiaoyong Du. 2023. Homomorphic Compression: Making Text Processing on Compression Unlimited. *Proc. ACM Manag. Data* 1, 4, Article 271 (Dec. 2023), 28 pages. doi:10.1145/3626765

[26] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: a cloud native vector database management system. *Proceedings ofthe VLDB Endowment* 15 (2022), 3548–3561. doi:10.14778/3554821.3554843

[27] Kiana Hajebi, Yasin Abbasi-Yadkori, Hossein Shahbazi, and Hong Zhang. 2011. Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph. In *IJCAI 2011, Proceedings ofthe 22nd International Joint Conference on Artificial Intelligence*. Barcelona, Catalonia, Spain. doi:10.5591/978-1-57735-516-8/IJCAI11-222 AAAI Press, 1312–1317..

[28] P. Jain, B. Kulis, and K. Grauman. 2008-06. Fast image search for learned metrics. In *2008 IEEE Conference on Computer Vision and Pattern Recognition.* 1–8,.

[29] Virajith Jalaparti, Peter Bodik, Srikanth Kandula, Ishai Menache, Mikhail Rybalkin, and Chenyu Yan. 2013. Speeding up distributed request-response workflows. *ACM SIG-COMM Computer Communication Review* 43, 4 (2013), 219–230.

[30] 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. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf

[31] 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.

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

[33] 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 (CVPR*. 2321–2328,.

[34] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM international conference on Information and knowledge management.* 535–544,.

[35] Brian Kulis and Kristen Grauman. 2009. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on.* IEEE, Elsevier Science Publishers B, V., NLD, 2130–2137.

[36] Daniel LeJeune, Richard G. Baraniuk, and Reinhard Heckel. 2019. Adaptive Estimation for Approximate k-Nearest-Neighbor Computations.

[37] Haitao Li, Qingyao Ai, Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Zheng Liu, and Zhao Cao. 2023. Constructing Tree-based Index for Efficient and Effective Dense Retrieval *(SIGIR '23).* Association for Computing Machinery, New York, NY, USA, 131–140. doi:10.1145/3539618.3591651

[38] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. 2017. Fexipro: fast and exact inner product retrieval in recommender systems. In *Proceedings of the 2017 ACM International Conference on Management of Data.* 835–850,.

[39] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyuan Ni, Ning Wang, and Yuan Chen. 2018. The Design and Implementation ofa Real Time Visual Search System on JD E-Commerce Platform. In *Proceedings ofthe 19th International Middleware Conference Industry (Rennes, France.* Association for Computing Machinery, New York, NY, USA, 9–16. doi:10.1145/3284028.328403

[40] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, XiaoMing Wu, and Qianli Ma. 2021. Embedding-Based Product Retrieval in Taobao Search. In *Proceedings ofthe 27th ACMSIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21.* Association for Computing Machinery, New York, NY, USA, 3181–3189. doi:10.1145/3447548.3467101

[41] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. Lightrec: A memory and search-efficient recommender system. In *Proceedings of The Web Conference 2020.* 695–705,.

[42] Ting Liu, Andrew W. Moore, Alexander Gray, and Ke Yang. December 13-18, 2004. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004.* Vancouver, British Columbia, Canada, 825–832,.

[43] Zihan Liu, Wentao Ni, Jingwen Leng, Yu Feng, Cong Guo, Quan Chen, Chao Li, Minyi Guo, and Yuhao Zhu. 2024. JUNO: Optimizing High-Dimensional Approximate Nearest Neighbour Search with Sparsity-Aware Algorithm and Ray-Tracing Core Mapping. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir (Eds.). ACM, 549–565. doi:10.1145/3620665.3640360

[44] Yury Malkov. 2020. Hnswlib: Fast approximate nearest neighbor search. https://github.com/nmslib/hnswlib. Accessed: 2024-10-15.

[45] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (apr 2020), 824–836. doi:10.1109/TPAMI.2018.2889473

[46] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. 2019. Howto100m: Learning a text-video embedding by watching hundred million narrated video clips. In *Proceedings ofthe IEEE/CVF International Conference on Computer Vision.* 2630–2640,.

[47] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL.

[48] Yadong Mu and Shuicheng Yan. 2010. Non-Metric Locality-Sensitive Hashing. In *AAAI.* AAAI Press, 539–544.

[49] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbour Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 11 (2014), 2227–2240.

[50] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, and Paul Saab. 2013. Scaling memcache at facebook. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13.* 385–398,.

[51] John Paparrizos, Ikraduya Edian, Chunwei Liu, Aaron J. Elmore, and Michael J. Franklin. 2022. Fast Adaptive Similarity Search through Variance-Aware Quantization. In *2022 IEEE 38th International Conference on Data Engineering (ICDE).* 2969–2983. doi:10.1109/ICDE53745.2022.00268

[52] Mattis Paulin, Matthijs Douze, Zaid Harchaoui, Julien Mairal, Florent Perronin, and Cordelia Schmid. 2015. Local convolutional features with unsupervised training for image retrieval. In *Proceedings ofthe IEEE international conference on computer vision.* 91–99,.

[53] Zhen Peng, Minjia Zhang, Kai Li, Ruoming Jin, and Bin Ren. 2023. iQAN: Fast and Accurate Vector Search with Efficient Intra-Query Parallelism on Multi-Core Architectures. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, PPoPP 2023, Montreal, QC, Canada, 25 February 2023 - 1 March 2023*, Maryam Mehri Dehnavi, Milind Kulkarni, and Sriram Krishnamoorthy (Eds.). ACM, 313–328. doi:10.1145/3572848.3577527

[54] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. *EMNLP* 14 (2014), 1532–1543. doi:10.3115/v1/D14-1162

[55] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. 2017. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *Int. J. Autom. Comput.* 14, 5 (oct 2017), 503–519. doi:10.1007/s11633-017-1054-2

[56] Jie Ren, Minjia Zhang, and Dong Li. 2020. HM-ANN: Efficient BillionPoint Nearest Neighbor Search on Heterogeneous Memory. *Proceedings ofthe 34th International Conference on Neural Information Processing Systems* 895 (2020), 20.

[57] Christian Rieger and Holger Wendland. 2024. On the Approximability and Curse of Dimensionality of Certain Classes of High-Dimensional Functions. *SIAM J. Numer. Anal.* 62, 2 (2024), 842–871. doi:10.1137/22M1525193

[58] Amaia Salvador, Nicholas Hynes, Yusuf Aytar, Javier Marin, Ferda Ofli, Ingmar Weber, and Antonio Torralba. 2017. Learning cross-modal embeddings for cooking recipes and food images. In *Proceedings ofthe IEEE conference on computer vision and pattern recognition.* 3020–3028,.

[59] Erich Schubert. 2021. A triangle inequality for cosine similarity. In *International Conference on Similarity Search and Applications.* Springer, 32–44.

[60] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. 2013. Streaming Similarity Search over One Billion Tweets Using Parallel Locality-Sensitive Hashing. *Proc. VLDB Endow* 6, 14 (2013). doi:10.14778/2556549.2556574

[61] Xilin Tang, Feng Zhang, Shuhao Zhang, Yani Liu, Bingsheng He, Bingsheng He, Xiaoyong Du, and Xiaoyong Du. 2024. Enabling Adaptive Sampling for Intra-Window Join: Simultaneously Optimizing Quantity and Quality. *Proc. ACM Manag. Data* 2, 4, Article 198 (Sept. 2024), 31 pages. doi:10.1145/3677134

[62] Godfried T. Toussaint. 1980. The relative neighbourhood graph of a finite planar set. *Pattern recognition* 12, 4 (1980), 261–268.

[63] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-nn graph construction for visual descriptors. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, IEEE Computer Society, USA.

[64] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian Sheng Hua. 2014. Trinary-projection trees for approximate nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 2 (2014), 388–403.

[65] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua

Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings ofthe 2021 International Conference on Management ofData (Virtual Event, China) (SIGMOD '21.* Association for Computing Machinery, New York, NY, USA, 2614–2627. doi:10.1145/3448016.3457550

[66] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. 2018. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2018), 769–790.

[67] Xueyi Wang. 2011. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. In *The 2011 International Joint Conference on Neural Networks.* 1293–1299. doi:10.1109/IJCNN.2011.6033373

[68] Yuke Wang, Boyuan Feng, Gushu Li, Georgios Tzimpragos, Lei Deng, Yuan Xie, and Yufei Ding. 2021. TiAcc: Triangle-inequality based Hardware Accelerator for K-means on FPGAs. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid).* 133–142. doi:10.1109/CCGrid51090.2021.00023

[69] Yuke Wang, Zhaorui Zeng, Boyuan Feng, Lei Deng, and Yufei Ding. 2019. KPynq: A Work-Efficient Triangle-Inequality Based K-Means on FPGA. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* 320–320. doi:10.1109/FCCM.2019.00061

[70] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: A Hybrid Analytical Engine towards Query Fusion for Structured and Unstructured Data. *Proc. VLDB Endow* 13, 12 (2020), 3152–3165. doi:10.14778/3415478.3415541

[71] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *Advances in neural information processing systems.* 1753–1760,.

[72] Shitao Xiao, Zheng Liu, Weihao Han, Jianjin Zhang, Yingxia Shao, Defu Lian, Chaozhuo Li, Hao Sun, Denvy Deng, and Liangjie Zhang. 2022. Progressively optimized bi-granular document representation for scalable embedding based retrieval. In *Proceedings ofthe ACMWeb Conference 2022.* 286–296,.

[73] Qian Xu, Juan Yang, Feng Zhang, Zheng Chen, Jiawei Guan, Kang Chen, Ju Fan, Youren Shen, Ke Yang, Yu Zhang, and Xiaoyong Du. 2024. Improving Graph Compression for Efficient Resource-Constrained Graph Analytics. *Proc. VLDB Endow.* 17, 9 (Aug. 2024), 2212–2226. doi:10.14778/3665844.3665852

[74] Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, Peng Cheng, and Mao Yang. 2023. SPFresh: Incremental In-Place Update for Billion-Scale Vector Search. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (Eds.). ACM, 545–561. doi:10.1145/3600006.3613166

[75] Shulin Zeng, Zhenhua Zhu, Jun Liu, Haoyu Zhang, Guohao Dai, Zixuan Zhou, Shuangchen Li, Xuefei Ning, Yuan Xie, Huazhong Yang, and Yu Wang. 2023. DF-GAS: a Distributed FPGA-as-a-Service Architecture towards Billion-Scale Graph-based Approximate Nearest Neighbor Search. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23).* Association for Computing Machinery, New York, NY, USA, 283–296. doi:10.1145/3613424.3614292

[76] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. 2023. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023*, Roxana Geambasu and Ed Nightingale (Eds.). USENIX Association, 377–395. https://www.usenix.org/conference/osdi23/presentation/zhang-qianxi

[77] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite quantization for approximate nearest neighbor search. In *Proceedings ofthe 31th International Conference on Machine Learning (ICML*, Vol. 32. 838–846,.

Received July 2024; revised September 2024; accepted November 2024