Fast, Approximate Vector Queries on Very Large Unstructured Datasets

Zili Zhang and Chao Jin, *Peking University;* Linpeng Tang, *Moqi;*
Xuanzhe Liu and Xin Jin, *Peking University*

https://www.usenix.org/conference/nsdi23/presentation/zhang-zili

This paper is included in the
Proceedings of the 20th USENIX Symposium on
Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

Open access to the Proceedings of the
20th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Fast, Approximate Vector Queries on Very Large Unstructured Datasets

Zili Zhang* Chao Jin* Linpeng Tang† Xuanzhe Liu* Xin Jin*

*Peking University †Moqi

Abstract

The breakthroughs in deep learning enable unstructured data to be represented as high-dimensional feature vectors for serving a wide range of applications. Processing vector queries (i.e., finding the nearest neighbor vectors for an input vector) for large unstructured datasets (with billions of items) is challenging, especially for applications with strict service level objectives (SLOs). Existing solutions trade query accuracy for latency, but without any guarantees, causing SLO violations.

This paper presents Auncel, a vector query engine for large unstructured datasets that provides bounded query errors and bounded query latencies. The core idea of Auncel is to exploit local geometric properties of individual query vectors to build a precise error-latency profile (ELP) for each query. This profile enables Auncel to sample the right amount of data to process a given query while satisfying its error or latency requirements. Auncel is a distributed solution that can scale out with multiple workers. We evaluate Auncel with a variety of benchmarking datasets. The experimental results show that Auncel outperforms state-of-the-art approximate solutions by up to 10× on query latency with the same error bound ($\leq$ 10%). In particular, Auncel only takes 25 ms to process a vector query on the DEEP1B dataset that contains one billion items with four c5.metal EC2 instances.

1 Introduction

Vector query engines for unstructured datasets (e.g., images, videos and texts) are a key building block for modern applications including recommendation [1–4], recognition [5–8] and biological information retrieval [9–11]. This is enabled by the breakthroughs in deep learning [12] that allow unstructured data to be represented as high-dimensional feature vectors. A vector query is to find the top-k nearest neighbor vectors in a dataset for an input vector.

With the explosive growth of unstructured data [13, 14], a central challenge for vector query processing is to satisfy strict service level objectives (SLOs) for applications on large unstructured datasets that contain millions and even billions of items. For instance, a face recognition task is to match a human face from an input image against a database of faces. With deep convolutional neural networks [6], each face image is converted into an embedding vector. Consequently, the recognition task becomes a top-k nearest neighbor (KNN) search problem, i.e., finding the nearest neighbor vector of the query vector among the database vectors. The person corresponding to the nearest neighbor vector is the recognition result. Performing exact KNN search (e.g., through pairwise comparison between query vector and each stored vector) is costly in terms of computation resources, and more importantly, is hard to achieve low query latency. As a result, approximate top-k nearest neighbor (ANN) search [15–18] is widely used by vector query engines to tradeoff query accuracy for latency. The basic idea of ANN search is to sample a subset of the dataset for finding the top-k, and the sampling size affects the query accuracy and latency.

A key requirement for approximate query processing is to provide *performance guarantees* in order to meet SLOs [19–21]. Performance guarantees are defined in terms of error bounds (e.g., $\leq$10% query error) or latency bounds (e.g., $\leq$25 ms query latency). Existing systems [22–29] exploit various ANN algorithms [15–18] and system optimizations to optimize query accuracy and latency. However, these systems do not provide any performance guarantees.

Faiss [22] and AnalyticDB-V [24] are widely-used open-source and commercial vector query engines, respectively. Unfortunately, they do not provide any performance (error or latency) bounds. They build a profile to map query errors to sampling sizes for a given dataset. It is possible to leverage the profile to pick an appropriate sampling size to meet an error bound. But the problem is that the profile is *query-agnostic*: it ignores the characteristics of individual query vectors, and uses a *fixed* sampling size for all query vectors under the same error bound. Consequently, the sampling size is too pessimistic—the maximum sampling size among all query vectors has to be used to meet the error bound. This leads to excessive redundant computation.

Learned Adaptive Early Termination (LAET) [30] is a recent work that leverages machine learning to optimize vector query processing. It trains a gradient boosting decision tree model to predict when to stop searching for a given query in order to reduce query latency. It focuses on average query accuracy, and does not provide any error or latency bounds. LAET includes a heuristic to adapt the decision tree model by multiplying a hyperparameter to the prediction result to meet

a given error bound. But the model treats the entire structured ANN index as a blackbox, which performs poorly on query latency for bounded errors.

More importantly, existing systems focus on a *single-node* setup and use a *single* worker to process each query. Distributed processing is critical for vector queries over large unstructured datasets with billions of items. Conceivably, one can replicate a dataset to multiple workers, and process multiple queries in parallel—one query by each worker. This naive solution has high memory footprint for billion-scale datasets (e.g., 360 GB for DEEP1B [31] dataset). Moreover, it cannot reduce query latency with more workers, making it hard to achieve latency bounds for billion-scale datasets.

We present Auncel, a vector query engine for large unstructured datasets with performance guarantees. Different from existing systems, Auncel allows users to specify an error bound or latency bound for an input vector. The core of Auncel is a query-aware and error-aware *error-latency profile* (ELP) that enables Auncel to minimize the query latency for an error bound and maximize the query accuracy for a latency bound. Auncel is a distributed solution that can reduce the query latency with more workers. To the best of our knowledge, Auncel is the first distributed vector engine that provides bounded errors and bounded latencies.

There are two primary challenges in realizing Auncel. The first challenge is to decide the appropriate sampling size for an individual query vector under a particular error or latency bound. Auncel uses a *whitebox* approach that exploits the geometric properties in the high-dimensional space to explicitly model the relationship between sampling sizes and query errors. This enables Auncel to build more accurate ELPs than existing query-agnostic or blackbox approaches. Auncel immediately terminates the search process when the error bound can be guaranteed based on the ELP to minimize query latency. In terms of the latency bound, Auncel exploits the nature of vector query processing and uses a runtime approach to maximize query accuracy.

The second challenge is to scale Auncel out to multiple workers in order to reduce query latency. A natural approach is to shard a dataset among workers and aggregate workers' results for query processing. The nuance is to correctly set the local error bound for each worker. Naively setting the local error bound to be the target error bound would magnify the global error after aggregation. Auncel applies probability theory to calibrate the local error bound for each worker in order to bound the global error. We theoretically prove that Auncel is able to bound the global error with high probability.

We implement a prototype of Auncel, and extensively evaluate it with a variety of benchmarking datasets. The results show that Auncel outperforms Faiss [22] by 1.3–10× and LAET [30] by 1.4–3.6× on the single-node setup. For the distributed setup, Auncel is able to process a vector query under 25 ms for the DEEP1B dataset which contains one billion vectors of 96 dimensions with 128 workers (using four c5.metal EC2 instances).

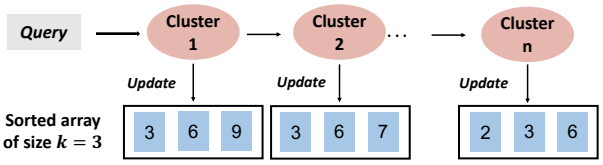


Figure 1: IVF workflow.

In summary, we make the following contributions.

- We present Auncel, to the best of our knowledge, the first distributed vector query engine that provides bounded query errors and bounded query latencies.
- We propose a whitebox approach that leverages high-dimensional geometry to build accurate ELPs, and apply probability theory to calibrate each worker to scale out.
- We implement an Auncel prototype. The evaluation shows that Auncel outperforms Faiss by up to 10× and LAET by up to 3.6×, and processes a vector query within 25 ms for billion-scale datasets with 128 workers.

2 Background and Motivation

In this section, we begin by introducing the background of vector queries on unstructured datasets. We then describe current solutions and their limitations to support vector queries on large unstructured datasets, which motivates the design of Auncel. Finally, we describe the challenge to scale out vector query processing.

2.1 Vector Queries on Unstructured Datasets

The common practice for managing and querying unstructured datasets is to use deep neural networks (DNNs) to process each item and represent each item as a high-dimensional feature vector [32–34]. A vector query on an unstructured dataset is to find the top-k vectors in the dataset that are most similar to the query vector. The most widely-used similarity metrics between two vectors are Euclidean distance and Angular distance. KNN search returns the top-k most similar vectors (i.e., nearest neighbors), and ANN search returns the approximate top-k nearest neighbors. KNN becomes impractical for large datasets with millions or billions of items due to long query latency. ANN trades off accuracy for latency, and is the de facto solution for vector query processing on large unstructured datasets. Another reason for the wide adoption of ANN is that it is unnecessary to output the exact top-k items for many vector query processing tasks, as DNN models themselves are not perfect when generating these vectors.

The basic idea of ANN search is to use an indexing structure to sample a subset of the dataset to find the top-k neighbors. Inverted file index (IVF) [15, 25, 35] is a state-of-the-art ANN algorithm. While IVF has many variants, it has the following general workflow. It trains a list of cluster centroids by k-means clustering [36] offline. These cluster centroids form the index of the dataset; each vector is assigned to the closest

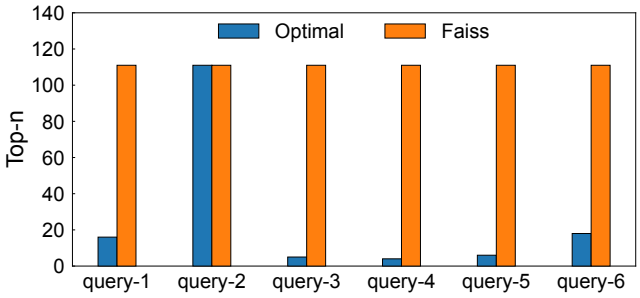


Figure 2: Redundant computation in Faiss.

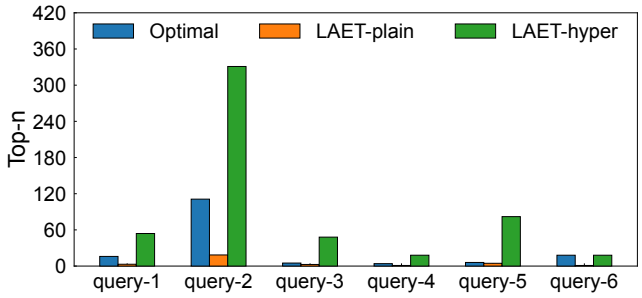


Figure 3: Redundant computation in LAET.

cluster. Given a query vector, IVF first computes the distance between each centroid and the query vector. Then it chooses the top-n nearest centroids, and processes the corresponding clusters of these centroids one by one. It maintains a sorted array of size k, and updates the array after performing pairwise distance calculation in each cluster as Figure 1. The k vectors remained in the array at the end are returned as the query result. The vectors in the array are called *intermediate result* during the processing. In this figure, each value in the array represents the vector's index in the ground truth result of exact search. The vectors in the top-n clusters are the sampling vectors. n determines the sampling size, and thus controls the tradeoff between accuracy and latency.

2.2 Bounding Performance for Vector Queries

Providing bounded performance for query processing is a key requirement for meeting SLOs of applications [19–21]. There are two typical types of performance bounds: error bounds and latency bounds. The query processing engine is expected to minimize query latency when given an error bound, and maximize query accuracy when given a latency bound.

Limitations of existing solutions. Existing systems [22–29] do not provide bounded performance, and leave the choice of the sampling size to users. While it is possible to adapt the mechanisms in existing systems to provide bounded performance, simply doing so yields undesirable results. Faiss [22] and AnalyticDB-V [24] build a profile by sampling some queries to map query errors to different n values (exponential power of two in practice to save the map building time) after building an IVF index for a dataset. To guarantee bounded errors, they use the n whose worst-case error is no bigger than the bound. This pessimistic choice of n has poor performance, because it is *query-agnostic*. It ignores the characteristics of individual queries and uses a fixed n for all queries under a given error bound. Some queries may use a smaller n (and thus achieve better latency) without violating the error bound.

To illustrate the problem, we randomly select six query vectors in DEEP10M [31] and assign them the same error bound (10%). In Figure 2, the optimal bars are the minimal values of n to reach the error bound for each query vector, and they are calculated through grid search of parameter n for the six queries respectively. Since Faiss uses the same value of n for all queries, the value is dominated by query-2; other queries do not need such a large n to meet the error bound. A larger n means searching more clusters, i.e., longer query latency. The naive solution of using a query-agnostic fixed value for n has a 10× gap from the optimal for some queries in this example.

To alleviate this problem, LAET [30] leverages machine learning to adaptively decide n among different query vectors. However, LAET is designed to reduce latency under *average* query error and is incapable to guarantee *bounded* error. It trains a decision tree model with LightBGM [37] and treats IVF index as a blackbox. The model takes the query vector, the intermediate result and some features of the clusters as input and outputs n for a given query. Due to the overhead of running a machine learning model for each query, LAET cannot use complex models and it also simplifies the model input. For instance, it only considers a small portion of intermediate result and cluster centroids. Therefore, the model cannot accurately predict n with blackbox fitting. To guarantee an error bound, LAET includes a heuristic to adapt the model by multiplying a hyperparameter to the prediction result—a tighter bound requires a larger hyperparameter. However, applying such a hyperparameter to all queries given the same error bound induces severely redundant computation with the inaccurate model. This is because this inaccurate blackbox model needs a very large hyperparameter to guarantee the error bound for all queries, but most queries only require a small one. Consequently, the values of n generated by LAET are also far from the optimal. Besides, tuning the hyperparameter for different error bounds is error-agnostic.

We continue with the previous DEEP10M example to show the problem of LAET. As shown in Figure 3, the bars of LAET-plain are lower than those of the optimal, indicating that the plain LAET solution with blackbox fitting is inaccurate to predict top-n and cannot provide bounded errors. LAET-hyper, which is LAET with the aforementioned heuristic, sets the hyperparameter large enough to ensure the bounded error for all the six queries. The hyperparameter is dominated by query-6; other queries can use a smaller hyperparameter, i.e., just enough to match the optimal. Therefore, LAET has the similar problem as Faiss and AnalyticDB-V. The inaccurate blackbox model introduces a 5× gap from the optimal for some queries in this example.

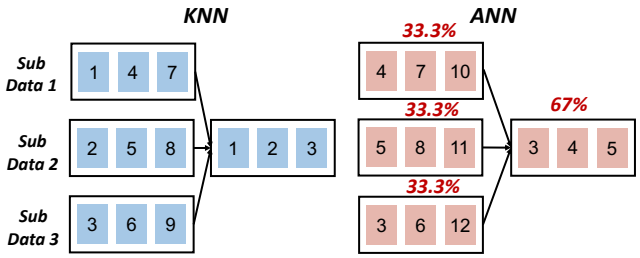


Figure 4: Problem when scaling error bounds.

Opportunity: Geometric structures in ANN indexes. The key to address the problem is building an accurate, lightweight ELP. Our core idea is to exploit the geometric structure and computation pattern of ANN indexes to establish the relationship between sampling sizes and query errors with high-dimensional geometry. The geometric intuition is that a query vector needs a large value of n if the vector is located at the *boundary* of clusters. Similarly, searching a small number of clusters, i.e., a small value of n, is sufficient, if the query vector falls close to a centroid. Thus, we can formulate the entire search procedure with geometric formulas, and use whitebox approach to explicitly model the relationship between sampling sizes and query errors.

2.3 Applying to Distributed Settings

Existing systems focus on a single-node setup and use a single worker to process each query. Replicating the dataset to each worker and processing multiple queries in parallel only increases throughput. It does not help with per-query latency, and has high memory footprint for each worker, both of which are undesirable for billion-scale datasets. A common approach is sharding, i.e., partitioning the dataset among multiple workers. Each worker finds the local top-k in its own shard (i.e., a *map* operation), and then a leader aggregates the local results to the global top-k (i.e., a *reduce operation*). This works well for exact search (KNN), as the aggregated result is identical to the ground truth. However, for approximate search (ANN), the error of the aggregated result is not bounded, even if the error of the local top-k on each worker is bounded.

To see why this is the case, consider the example in Figure 4. The example includes three workers, the value of k is 3, and the error bound is 35%. We show the local top-k at each worker and the aggregated global top-k. Each value represents the corresponding top-k vector's index in the global ground truth result of exact search. The results of KNN is on the left and that of ANN is on the right. In KNN, each worker returns the exact local top-k, and the aggregated top-k vectors are the true top-k (i.e., the ground truth). In ANN, the error of the local top-k at each worker is 33.3%, which satisfies the error bound. However, after aggregating the local top-k, the error of the global top-k is 66.7%; only one vector (with index 3) is in the true global top-k vectors.

To address the problem, we need to calibrate the local error bounds when finding the local top-k at each worker; we cannot directly use the global error bound. We apply probability theory to calibrate the local error bounds to ensure that the global error is bounded with high probability.

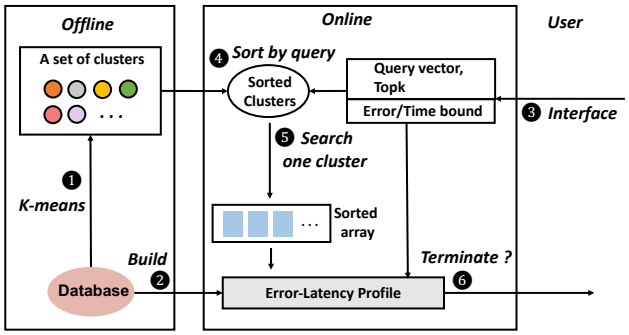


Figure 5: Auncel architecture.

3 Auncel Overview

We present Auncel, a vector query processing engine for large unstructured datasets with performance guarantees. Auncel exploits the geometric properties of ANN index structures to build accurate, lightweight ELPs. Such ELPs enable Auncel to sample just enough data to answer vector queries within their error or latency bounds. To scale out vector query processing for large datasets with billions of items, Auncel adopts map-reduce style dataflow operations, and applies probability theory to calibrate the local error bounds at each worker. Figure 5 shows the overall architecture of Auncel. We provide a brief overview of Auncel in this section.

User interface. Auncel allows users to tradeoff between accuracy and latency with user-defined performance bounds ❸. Specially, a user can specify an error bound or a latency bound for a query vector and a value of k (i.e., how many nearest neighbors to return) as follows.

- **Error bound ε.** The user specifies an error bound ε, and Auncel returns a result within ε error as soon as possible.
- **Time bound t.** The user gives a time bound t, and Auncel returns the most accurate result within t time.

Offline. Similar to all query processing engines for unstructured datasets, Auncel first uses IVF to build an ANN index for a given dataset offline ❶. IVF divides the vectors in the database into a few clusters with k-means clustering. It maintains the centroids of each cluster; each vector in the dataset is assigned to the closest cluster. In addition to the ANN index, Auncel samples synthetic or example queries to build an ELP for the dataset ❷. ELP building techniques include fitting geometric formulas and substituting some complex operators with pre-calculated key-value pairs to reduce overhead.

Online. At runtime, vector queries are issued to Auncel for processing. Each query includes a vector, a value of k, and an error/time bound. To process an incoming query, Auncel first evaluates the distance between the query vector and each

Symbol	Description
q	Query vector
t	Latency bound
ε	Error bound
l	Number of clusters
C_i	The i_{th} closest cluster to q (a hyper polyhedron)
ε_i	Error after processing $\{C_1 ... C_i\}$
S	Set of database vectors
S_i	Intermediate result after processing $\{C_1 ... C_i\}$
S_{gt}	Ground truth result of exact search ($S_{gt} = S_l$)
j^*	$\lvert S_i \cap S_{gt} \rvert$ after processing $\{C_1 ... C_i\}$
$\varphi_i(j)$	Scaling factor of the j_{th} element in S_i
$\lambda_i(j)$	Distance between q and the j_{th} element in S_i
$B(r)$	Sphere (Ball) with center q and radius r
$P_j(m)$	$B(\lambda_i(j)) \cap C_m$
$N_j(m)$	Number of the vectors within $\bigcup_{\eta=1}^{m} P_j(\eta)$
$V(G)$	Volume of geometric body G

Table 1: Key notations in problem formulation.

centroid, and then sorts the clusters by distance in ascending order ❹. According to the sorted clusters, Auncel performs pairwise distance calculation between the query vector and each stored vector cluster by cluster, and updates the sorted array (i.e., the intermediate result) ❺. After processing each cluster, Auncel uses the intermediate result and the centroids to predict the current error based on the ELP ❻. If the error or time bound can be satisfied, Auncel terminates the search process in ❺, and returns the vectors in the array as the result.

4 Auncel Design

In this section, we present the design of Auncel. We first describe the problem formulation (§4.1) and the key ideas (§4.2). Then we show how to build error profiles (§4.3) and latency profiles (§4.4). Finally, we describe how to apply our solution to distributed settings (§4.5). Some key notations in the design are listed in Table 1.

4.1 Problem Formulation

We first mathematically formulate the problem of vector query processing on unstructured datasets. Let $S = \{v_1, v_2, ...v_N\} \in \mathbb{R}^d$, where S is an unstructured dataset, N is the number of vectors in S, and v_i is a d-dimensional vector in S. Let $q \in \mathbb{R}^d$ be a query vector. Given a value $k \in \mathbb{N}_+$ and $k \leq N$, a vector query with q is to find the top-k nearest neighbors of q in S, according to a pairwise distance function $d(q, v_i)$. The distance function typically computes Euclidean distance or Angular distance between two vectors. The ground truth of the top-k nearest neighbors S_{gt} is obtained when searching in the entire dataset S. S_{gt} are often sorted according to $d(q, v)$.

$$S_{gt} = argmin^k_{v \in S} \quad d(q, v) \tag{1}$$

Finding the exact top-k nearest neighbors has high query latency for large datasets, which may violate latency SLOs

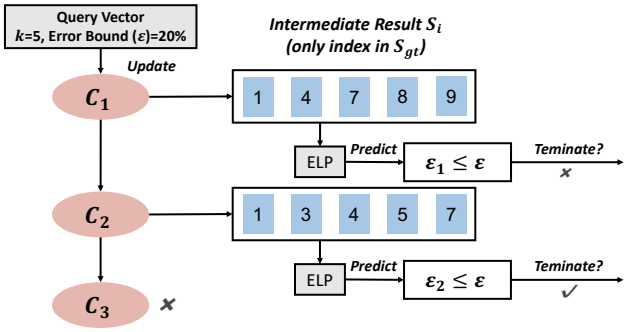


Figure 6: Example to show key idea and workflow of Auncel.

for applications. To trade query accuracy for query latency, a subset $S_a \subset S$ can be sampled to find the approximate top-k nearest neighbors S_r for lower latency.

$$S_r = argmin^k_{v \in S_a} \quad d(q, v). \tag{2}$$

The accuracy (recall) and error of S_r are defined as follows.

$$Accuracy \triangleq \frac{|S_{gt} \cap S_r|}{k} \tag{3}$$

$$Error \triangleq 1 - Accuracy \tag{4}$$

4.2 Workflow and Key Idea

Auncel allows users to specify an error bound or a latency bound for a query. When an error bound is given, Auncel minimizes query latency; when a latency bound is given, Auncel maximizes query accuracy. As we have described in §2, the basic approach for processing a vector query with an ANN index is to first compute the distances between the query vector and the centroids of the clusters in the index and then search cluster by cluster based on the ascending order of the distances. The key idea of Auncel is to build an accurate ELP so that after searching each cluster, Auncel can consult the ELP to decide whether to terminate the search.

Workflow of Auncel. We use a concrete example in Figure 6 to illustrate the workflow of Auncel. In this example, the ANN index partitions the dataset into three clusters. Each cluster (C_i) is a polyhedron in d-dimensional space. The value of k in top-k is 5, and the error bound is 20%. The three clusters are sorted in ascending order by their centroids' distances to the query vector. Auncel searches the three clusters based on the order one by one. The three clusters may have different numbers of vectors due to the imbalance property of k-means. It does not impact the error profile since Auncel terminates the search as soon as the current error is guaranteed. However, the imbalance property leads to inaccurate latency profile that we will discuss later in §4.4.

After processing C_1, Auncel updates the intermediate result (a sorted array) which represents the query vector's top-k

Algorithm 1 Error Profile

```
1: ε ← Error Bound, i ← 1
2: while i <= l do
3:     Perform search computation in cluster C_i
4:     geometric properties ← S_i , cluster centroids
5:     ε_i ← ELP (geometric properties)
6:     if ε_i ≤ ε then
7:         break
8:     i ← i + 1
9: Return intermediate result
```

nearest neighbors in C_1. Each element in the sorted array is a pair of a database vector's index and its distance to the query vector. For simplicity, the figure only shows the element's index in ground truth result. Auncel uses the intermediate result as the input of the ELP to predict the current error, and decides whether to terminate the search. The ground truth vectors have top-k indexes [1, 2, 3, 4, 5] in this example, which is obtained after searching all three clusters. As the intermediate result after searching C_1 is $[1,4,7,8,9]$, only two vectors in the intermediate result (the two with indexes 1 and 4) belong to the true top-k. Therefore, the current error is 60%, which is still above the error bound 20%. Auncel continues to search C_2. The intermediate result after searching C_2 contains four vectors in the true top-k (the four with indexes 1, 3, 4 and 5), and the current error decreases to 20%, which satisfies the error bound. Therefore, it is safe for Auncel to terminate the search and return the sorted array as the query result.

The workflow is summarized in Algorithm 1. Predicting the error with ELP in line 5 is the key to guarantee bounded errors and minimize query latency. If the predicted error is smaller than the actual error, the system terminates search too early, and fails to meet the error bound; if the predicted error is larger than the actual error, the system unnecessarily continues to search more clusters, which increases query latency. Thus, the main challenge is to build an accurate ELP to accurately predict the current error after searching each cluster.

Key idea of Auncel. To understand how Auncel addresses this challenge, consider the intermediate result after searching C_1. The first element in the intermediate result (1) is also the first element in ground truth (sorted array after searching all three clusters). But, the second element (4) is the fourth element in ground truth, and the the third element (7) is not in the top-k (k=5). We define the scaling factor of the j_{th} element in the intermediate result after processing C_i as follows, where $index_{gt}$ is the element's index in ground truth.

$$\varphi_i(j) \triangleq index_{gt}/j \quad (\geq 1) \tag{5}$$

If $\varphi_i(j)$ is known, then the current error of the intermediate result after searching C_i can be calculated. Specifically, we compute j^* such that

$$j^* = argmax_j\{j \cdot \varphi_i(j) \leq k\}. \tag{6}$$

The elements from 1 to j^* in the intermediate result belong to the true top-k. j^* is often calculated through binary search. The current error ε_i after processing C_i is

$$\varepsilon_i = 1 - j^*/k \tag{7}$$

For any j, $\varphi_i(j)$ converges to 1 when i increases to the number of clusters (l). Correspondingly, j^* converges to k and ε_i converges to 0. When searching the clusters one by one, Auncel terminates the process immediately when ε_i becomes no bigger than the error bound.

Therefore, we convert the problem of building an accurate ELP to accurately estimating $\varphi_i(j)$. We exploit the geometric properties of ANN indexes in high-dimensional space to estimate $\varphi_i(j)$. Since Auncel is designed to provide bounded errors, it is sufficient to estimate the upper bound of scaling factor $\varphi_i(j)$, which we describe next.

4.3 Handling Error Bounds

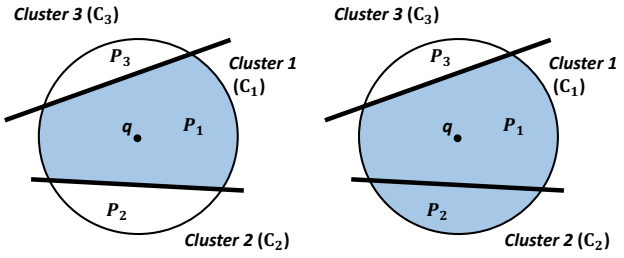
Auncel minimizes query latency under a given error bound using scaling factors in the i_{th} error prediction (φ_i). We perform a detailed analysis of $\varphi_i(j)$ from a geometric perspective, and design a formula to calculate the upper bound of $\varphi_i(j)$ under the two most prevalent distance metrics.

4.3.1 *Scaling Factor* under Euclidean Distance

We first focus on Euclidean distance, the most widely-used and intuitive distance metric, which measures the length of the line segment between two anchor vectors in geometry.

We have two key insights. Our first insight is to leverage the geometric structure of the IVF index. IVF shards the entire dataset into l clusters (C) by k-means clustering, and the clusters are sorted as $C = \{C_1, C_2 ... C_l\}$. Due to the rule of k-means, C_i is a d-dimensional *polyhedron* and the boundary between C_i and C_j is the $(d-1)$-dimensional mid-vertical plane of the line segment connecting the two clusters' centroids. Thus, we can divide the entire space into several parts based on these boundaries. Our second insight is to exploit the local geometric properties of q which belongs to C_1. The top-k ground truth vectors gather around q and form a sphere, $B(\lambda_l(k))$ with radius $\lambda_l(k)$ and center q in d-dimensional space. For any vector within $B(\lambda_l(k))$, it is a member of S_{gt}. For instance, if $B(\lambda_l(k))$ locates within C_1, it is sufficient to search in the first cluster to get ground truth.

The combination of these two insights allows us to compute $\varphi_i(j)$ with high-dimensional geometry. We know that the k ground truth vectors are distributed in sphere $B(\lambda_l(k))$, and the sphere is divided into many parts by the boundaries between C_1 and other clusters. For example, we have three clusters in total and query vector q in Figure 7. The sorted clusters are $\{C_1, C_2, C_3\}$, and all vectors within the sphere belong to S_{gt}. This sphere is cut into three parts—P_1, P_2, P_3—by the two boundaries, and N_i is the number of database vectors in the scope of $\bigcup\{P_1 ... P_i\}$ ($i = 1,2,3$). In Figure 7, the numbers of the vectors within the shaded areas are N_1 and N_2,



(a) Search in one cluster. (b) Search in two cluster.

Figure 7: Geometric demo for calculating scaling factor.

respectively. In Figure 7(a), scaling factor $\varphi_1(N_1) = \frac{k}{N_1}$ since the $N_{1_{th}}$ element in S_1 is the k_{th} element in S_{gt}. Figure 7(b) shows the scaling factor $\varphi_2(N_2) = \frac{k}{N_2}$. To extend to general settings, we define $P_j(m)$ as the intersection of $B(\lambda_i(j))$ and C_m after processing C_i. Consequently, $N_j(m)$ is the number of vectors within $\bigcup_{\eta=1}^{m} P_j(\eta)$ which gives that:

$$\varphi_i(j) = N_j(l)/N_j(i). \tag{8}$$

l is the number of clusters and i represents the current cluster.

We observe that the vectors of real-world datasets conform to *local uniform distribution*, and k in most query workloads is no bigger than 100, which means the vectors within the scope of $B(\lambda_l(k))$ nearly conform to uniform distribution. We provide a measurement in Appendix A.1 to confirm this observation. Let *den* represents the local density of $B(\lambda_l(k))$. We get that $N_j(m) \approx V(\bigcup_{\eta=1}^{m} P_j(\eta)) \times den$. Hence,

$$\varphi_i(j) = V(B(\lambda_i(j)))/V(\bigcup_{m=1}^{i} P_j(m)). \tag{9}$$

V is the volume function. Since $P_j(1)$ has different geometric meaning with $P_j(m)$ $(m \geq 2)$ (spherical cap) and is complex to calculate, we use the following inequation:

$$\begin{aligned}\varphi_i(j) &= \frac{1}{1 - \frac{V(\bigcup_{m=i+1}^{l} P_j(m))}{V(B(\lambda_i(j)))}} \\ &\leq \frac{1}{1 - \frac{\sum_{m=i+1}^{l} V(P_j(m))}{V(B(\lambda_i(j)))}} \leq \frac{1}{b - a \times U}.\end{aligned} \tag{10}$$

Different spherical caps (e.g., P_1, P_2 in Figure 7) may intersect with each other. The union of all spherical caps has a smaller volume than the sum of the volumes of all spherical caps, which leads to the first $\leq$. We apply $b - a \times U$ to substitute such complicated volume calculation in d-dimensional space, where a, b are parameters to fit offline and $a \times U$ is the geometric *upper bound* of the volume ratio. Appendix A.2 contains the detailed analysis of $a \times U$, and we conclude one of the upper bound functions is

$$U = \sum_{m=i+1}^{l} \arccos(x_m) \quad (0 \leq x_m \leq 1). \tag{11}$$

Algorithm 2 Latency Profile

```
t ← Time Bound, i ← 1
t_0 ← CurrentTime()
while i ≤ l do
    Perform search computation in cluster C_i
    t_c ← CurrentTime()
    if t_c − t_0 ≥ t − δ then
        break
    i ← i + 1
Return intermediate result
```

where $x_m = \frac{db_m}{\lambda_i(j)}$ and db_m is the distance between query vector and the boundary of C_1, C_m. We do not consider the circumstance when $x_m > 1$.

In Formula 11, the time complexity of calculating the upper bound of $\varphi_i(j)$ is $O(l)$ since $\arccos(x_m)$ only costs constant time. We use binary search to calculate j^* with Formula 6, which concludes that the time complexity to predict ε_i is $O(l \times \log(k))$ while the space complexity is $O(1)$.

4.3.2 *Scaling Factor* under Angular Distance

Another widely-used vector distance metric is Angular distance, which evaluates the angle between two anchor vectors. Its geometric meaning allows us to transform Angular distance into Euclidean distance. Specifically, we project all database vectors onto the unit sphere in d-dimensional space through *vector normalization*, while maintaining the Angular distance between any vectors. Thus, we substitute angle with the line segment on such unit sphere, and the theoretical analysis in §4.3.1 holds under Angular distance.

4.4 Handling Latency Bounds

Given a latency bound, Auncel maximizes query accuracy for a query vector. Conceivably, one can build a latency profile to capture the relationship between sampling sizes (i.e., the number of clusters to search) and query latencies. Then Auncel can consult the latency profile to get how many clusters to search based on a given latency bound. However, building such a profile is difficult, because the clusters have different sizes and the order of the set of clusters to search vary between different query vectors.

We design a *runtime* solution that exploits the *monotonicity* property of vector query processing to handle latency bounds, obviating the need of building a latency profile offline. Specifically, vector query processing searches the clusters in the ANN index one by one. The search is based on ascending order of the distances between the clusters' centroids and the query vector. The accuracy of the intermediate result increases when searching more clusters. As such, Auncel tracks the used time when searching the clusters, and terminates the search when the used time is close to the time bound. This ensures that Auncel uses as much time as possible in processing to maximize query accuracy. Algorithm 2 shows

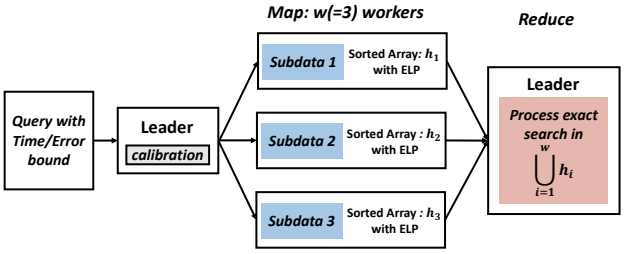


Figure 8: Auncel runtime with map-reduce.

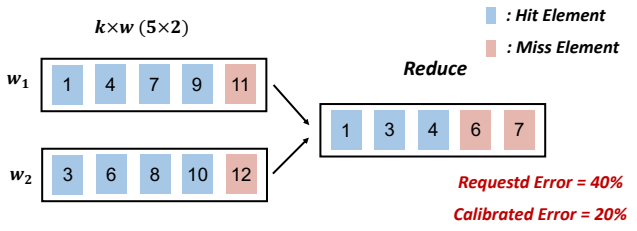


Figure 9: Error calibration of Auncel.

the pseudocode of processing vector queries under latency bounds. To guarantee the search terminates before the time bound, the algorithm stops the search when the used time is a (configurable) δ before the time bound. δ is influenced by the first cluster after Auncel terminates the search, i.e., C_i if the termination condition is triggered after processing C_{i-1}. The larger δ is required if C_i has more vectors and costs more time to process. To guarantee the latency bounds in any circumstances, δ is tuned according to the cluster with the largest number of vectors.

4.5 Applying to Distributed Settings

We emphasize that the distributed solutions mentioned in §2.3 either cannot reduce query latency (i.e., only improve throughput) or leads to error amplification. To scale out, Auncel uses map-reduce style processing to process vector queries with multiple workers while preserving the error and latency bound. Auncel divides distributed vector processing into two phases—a `map` phase and a `reduce` phase. As shown in Figure 8, Auncel randomly and uniformly shards the dataset into multiple partitions. Each worker owns one partition, and builds a local ANN index and a local ELP. One of the workers is elected as the leader, which controls the global query processing.

When a vector query with an error/time bound (which we call global bound) comes, the leader calibrates the bound to get an error/time bound to be used by each worker (which we call local bound). In the `map` phase, each worker uses its local ANN index and ELP to process the query on its own partition. In the `reduce` phase, the leader collects the local results from the workers, and performs exact top-k search on these collected results to produce the final result.

Calibration of error/latency bounds. As we have shown in §2.3, the local errors can be amplified in the `reduce` phase when the local results are aggregated to the final result. Directly using the global error bound as the local error bound in the `map` phase would cause the final error obtained by the `reduce` phase to be larger than the global error bound. To address this problem, we design an error bound calibration mechanism based on probability theory. The leader calibrates the local error bound before distributing the work to the workers in the `map` phase.

Figure 9 shows an example with two workers to illustrate the calibration mechanism. The global error bound ε is 40%, and the local error bound after calibration ε_c is 20%. Hit elements are the intersection of S_r and S_{gt}. The `map` phase is done by the workers individually, and the local error at each worker is 20%, which meets the local error bound ε_c. The `reduce` phase aggregates the local results and the global error is 40%. While the global error is bigger than 20%, it still meets the global error bound ($\varepsilon = 40\%$). The probability of the example in the figure is: $\binom{5}{3} * \binom{5}{5} / \binom{10}{8} = \frac{2}{9}$, where $\binom{n}{m}$ means the *combination number*. From classical models of probability, it means the probability of the random event that selects eight hit elements of all ten ground truth vectors and three elements are situated in the first five ground truth.

Formally, we define the number of hit elements of the first n elements in the ground truth as H_n. The worst case is $H_{k\times w} = k \times w \times (1-\varepsilon_c)$, which means each worker just meets the local error bound. The global error can be represented by $1 - H_k/k$. The probability of a random event $H_k = i$ is

$$P(H_k = i) = \frac{\binom{k}{i} \times \binom{k\times(w-1)}{k\times w\times(1-\varepsilon_c)-i}}{\binom{k\times w}{k\times w\times(1-\varepsilon_c)}}. \tag{12}$$

Thus, the probability of the random event to meet the global error bound is calculated by:

$$P(H_k \geq (k \times (1-\varepsilon))) = \sum_{i=k\times(1-\varepsilon)}^{k} P(H_k = i). \tag{13}$$

For instance in Figure 9, $P(H_k \geq (k \times (1-\varepsilon))) = P(3) + P(4) + P(5) = \frac{2}{9} + \frac{5}{9} + \frac{2}{9} = 1$, which means the calibrated local error bound (20%) can guarantee the global error (40%) under any circumstances. Auncel starts calibrating ε_c from ε and decreases ε_c by $\frac{1}{k}$ each time. We choose $\frac{1}{k}$ as the loop decrement because the error, which is defined in Formula 3, is a multiple of $\frac{1}{k}$. The probability to guarantee the global error bound ε is calculated through Formula 13. If the probability is greater than γ (99.9% used in our prototype), Auncel stops the calibration and distributes the ε_c to each worker. Therefore, the probability of failing to guarantee the error bound is less than 1-γ (<0.1% when $\gamma = 99.9\%$), i.e., Auncel guarantees the error bound with high probability. As for the convergence

time of the calibration algorithm, the calculation time of Formula 12 is constant since the values of combinatorial numbers are pre-calculated offline. The time complexity of Formula 13 is $O(k \times \varepsilon)$. Consequently, the convergence time of calibration is $O((k \times \varepsilon)^2)$. Since k in real query workloads is not large, the time to calibrate the error bound is relatively small.

Calibrating the latency bound is straightforward since the overhead of the `reduce` phase is negligible compared to the `map` phase. We slightly enlarge δ to include the reduce overhead as the latency calibration.

5 Implementation

We have implemented a system prototype of Auncel with ~3000 lines of code in C++ based on Faiss. We use Faiss for building IVF indexes and similarity search, and extend Faiss with building and applying ELPs for performance bounds. Because Faiss does not support distributed processing, we also implement data sharding, and map-reduce operations for distributed query processing. In theory, Auncel can be integrated with any vector query processing engines. We choose Faiss because it is a widely-used open-source vector query processing engine, and is adopted in production like Meta/Facebook. The code of Auncel is open-source and is publicly available at https://github.com/pkusys/Auncel.

ELP & Distributed setting. We implement the ELP component of Auncel with C++ Standard Library (STL) and Intel oneMKL [38]. ELP works in the query process and is treated as the process controller (monitor). For the distributed setting, Auncel spawns a worker process for each CPU core, and a server machine contains multiple workers. The entire dataset is sharded among the workers, and each worker processes queries on its own shard. Auncel randomly chooses a machine to spawn a leader process to receive the query and distribute the query with calibrated configurations (§4.5) to each worker. After all workers finishing their own query processing, the leader aggregates local results and returns the final results to the user. Each machine contains a daemon process that manages the local workers on the machine and communicates with the leader using TCP sockets. For leaders, Auncel handles query failures by re-executing the query. When receiving a local result from a worker, the leader creates a consistent backup of the result. Users are able to resume the query with the existing backup files, which is similar with the ideas of traditional primary-backup mechanisms [39, 40].

System optimizations. The ELP component imposes computation overhead to the system because of the complex geometric operations for high-dimensional vectors. We follow Faiss to implement Euclidean distance calculation, Angular distance calculation and vector normalization through SIMD instructions of oneMKL. Since SIMD is designed for vector operations such as inner-product and element-wise addition, it significantly reduces the computation overhead of the ELP component. In addition, we pre-calculate key-value pairs of some operations(e.g., arccos) offline and consult the key-value pairs online to improve the performance of these operations.

Dataset	Dimensions	Database Vectors	Query Vectors	Distance
SIFT10M [41]	128	10M	10K	Euclidean
DEEP10M [31]	96	10M	10K	Euclidean
DEEP1B [31]	96	1B	10K	Euclidean
GIST1M [42]	960	1M	1K	Euclidean
TEXT10M [31]	200	10M	10K	Angular

Table 2: Datasets used in the evaluation.

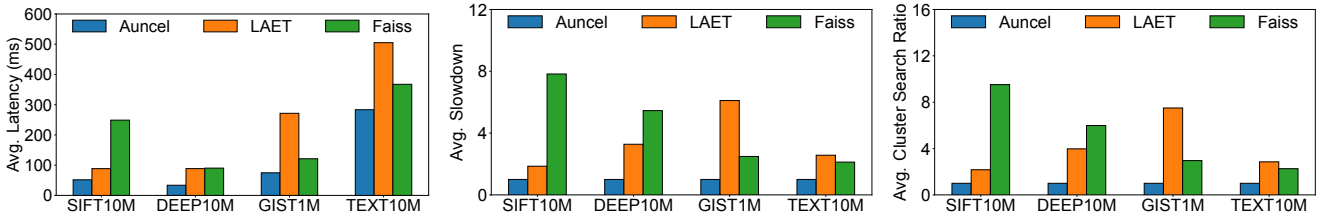
6 Evaluation

In this section, we empirically evaluate Auncel from the following aspects: (i) end-to-end performance improvement over state-of-the-art solutions; (ii) effectiveness of ELP; (iii) validation of the mathematical formulation; (iv) system overhead of Auncel; and (v) scalability. The summary of the experiments is as follows.

- Auncel outperforms LAET [30] and Faiss [22] by up to 3.6× and 10× on average query latency under the same error bound, respectively (§6.1).
- The ELPs built by Auncel are highly accurate across a range of datasets (§6.2).
- The mathematical formulation of Auncel fits well with real-world datasets (§6.3).
- The runtime overhead of Auncel is within 1%, and building an ELP offline can be done within ten minutes (§6.4).
- Auncel scales out near ideally, and only takes 25 ms to process a query on DEEP1B with 128 workers (§6.5).

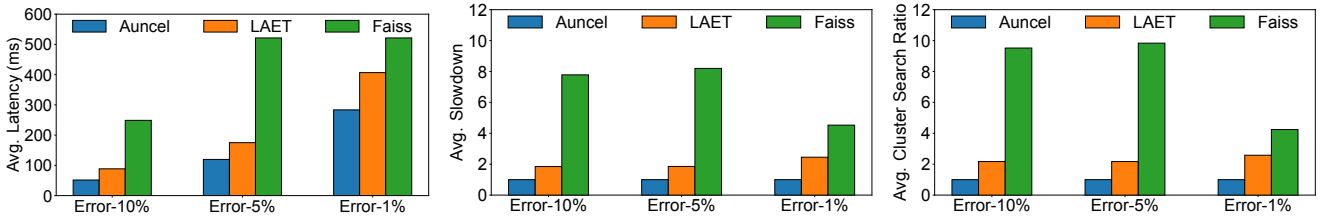
Setup. All experiments are conducted on AWS. We use two EC2 instance types, both configured with Ubuntu 18.04 LTS. For the single-node experiments, we use c5.4xlarge, which is configured with 16 vCPUs (Intel Xeon Platinum 8275CL) and 32 GB memory. For the scalability experiments, we use c5.metal, which is configured with 96 vCPUs (Intel Xeon Platinum 8275CL) and 192 GB memory. The reason of using c5.metal for the scalability experiments is the experiments aim to demonstrate the ability of Auncel to support very large datasets and we need large memory to host DEEP1B (nearly 400 GB memory footprint).

Datasets. Table 2 summarizes the preprocessed datasets used in our experiments. These datasets are widely-used benchmarking datasets for vector query processing in both academic and industry [43, 44]. Each dataset consists of database vectors, query vectors and ground truth. For each query vector, the ground truth records the indexes and distances of its top-100 neighbors. SIFT [41] is a dataset of local SIFT image descriptors [45] with ten million database vectors and ten thousand queries. DEEP [31] is a dataset of CNN [46] image embeddings with one billion database vectors and ten thousand queries. The single-node experiments only use ten million database vectors of DEEP, denoted by DEEP10M. The scala-



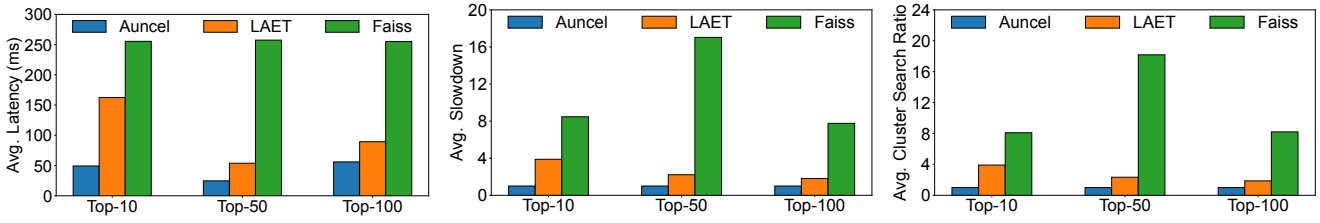
(a) Average latency. (b) Average slowdown of latency. (c) Average cluster search ratio.

Figure 10: Performance under different datasets.



(a) Average latency. (b) Average slowdown of latency. (c) Average cluster search ratio.

Figure 11: Performance under different error bounds.



(a) Average latency. (b) Average slowdown of latency. (c) Average cluster search ratio.

Figure 12: Performance under different values of k.

bility experiments use all database vectors of DEEP, denoted by DEEP1B. GIST [42] is a dataset of global color GIST descriptors [45] with one million database vectors and one thousand query vectors. TEXT [31] is a cross-model [47,48] dataset of texts and images, where the ten million database vectors and the ten thousand query vectors have different distributions in a shared representation space. The TEXT dataset adopts Angular distance as the distance metric, while the other three datasets use Euclidean distance.

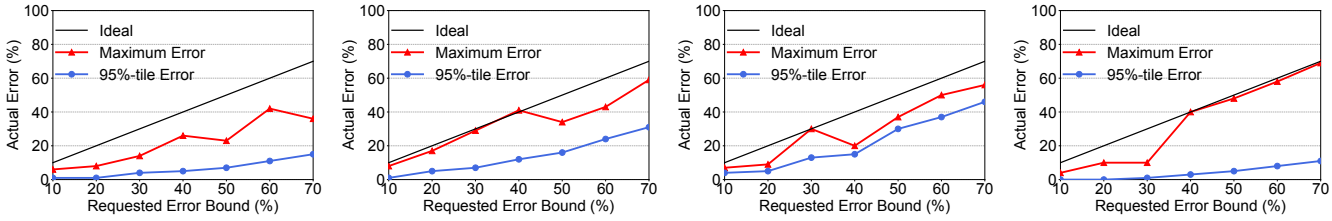
Baselines. We compare Auncel to two baselines.

- Faiss [22] is a widely-used solution for processing vector queries. It uses a fixed approach that picks the same sampling size for all queries under a given error bound.
- LAET [30] is a state-of-the-art solution that uses machine learning to adaptively determines search termination conditions for individual queries.

We emphasize that Faiss and LAET do not provide performance guarantees. To the best of our knowledge, Auncel is the first system that provides performance guarantees for vector query processing. In the experiments, we use the best configurations (e.g., the minimal n in the map for Faiss and the earliest search termination condition for LAET) to guarantee that all queries meet the given error bound. This allows us to fairly compare the query latency of Auncel, Faiss and LAET, while all three systems satisfy the given error bounds.

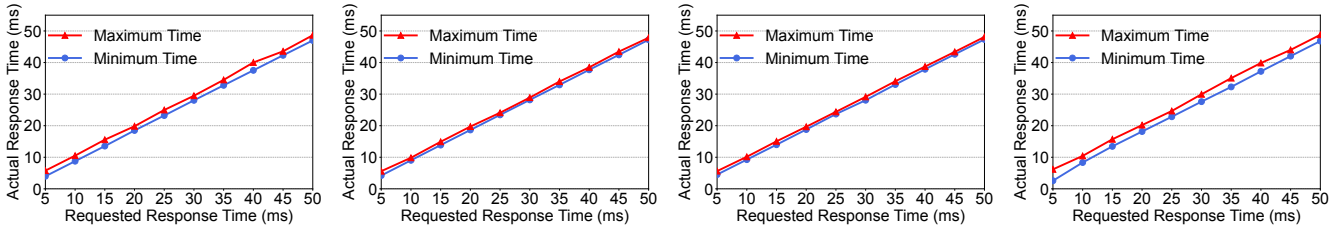
Note that it is necessary for all the three systems (Auncel, Faiss and LAET) to train their ELPs offline with some example queries. In the experiments, unless otherwise stated, we randomly split the query vectors into two parts of equal size, one for training and the other for testing. Because Auncel and LAET only perform search on the training queries once, the ELP building times of the two systems are almost the same. However, the grid search method of Faiss requires searching on the training queries for different top-n (exponential power of two in practice to save time), and the building time is tens of times longer than that of Auncel and LAET.

Metrics. We use average end-to-end query latency, $Ave(T_{system})$ as the main evaluation metric, where T_{system} represents the individual query latency of one of the three systems. In addition, we also report average slowdown of latency and average cluster search ratio. Average slowdown of latency is defined as $Ave(\frac{T_{baseline}}{T_{Auncel}})$. Average cluster search ratio is defined as the average ratio between the number of searched clusters by the baseline and that by Auncel, i.e., $Ave(\frac{N_{baseline}}{N_{Auncel}})$, where N represents the number of searched clusters when processing an individual query.



(a) SIFT10M. (b) DEEP10M. (c) GIST1M. (d) TEXT10M.

Figure 13: Effectiveness of error profiles.



(a) SIFT10M. (b) DEEP10M. (c) GIST1M. (d) TEXT10M.

Figure 14: Effectiveness of time profiles.

6.1 Overall Performance

We compare the end-to-end query latency between Auncel, Faiss and LAET under the same error bound. Auncel outperforms Faiss and LAET under different datasets, different error bounds, and different values of k.

Performance under different datasets. We compare Auncel against the baselines on SIFT10M, DEEP10M, GIST1M and TEXT10M. We use one c5.4xlarge EC2 instance. The error bound is 10% and k in top-k is 100 (i.e., returning top-100 nearest neighbors for a query). The results are shown in Figure 10, which we summarize as follows.

- Auncel achieves 1.8–3.6× lower average end-to-end latency than LAET and 1.3–4.8× lower of that than Faiss under the same error bound. This is because Auncel adaptively and accurately profiles the relationship between errors and sampling sizes, which allows Auncel to sample fewer clusters to generate query results.
- Compared to the gap of average latency between Auncel and the baselines, the gap of average slowdown is larger. This is because Auncel co-adapts individual queries and the error bounds, while the two baselines fail to adapt queries and error bounds at the same time.
- Auncel outperforms the two baselines by up to 9.6× in average cluster search ratio. This means Auncel can reduce the search cost by up to 9.6× in average while meeting the requested error bound, due to the use of the geometric properties when building ELPs.
- Auncel significantly outperforms Faiss and LAET on all the four datasets, which have different characteristics of data and different metrics of distance (Euclidean and Angular).

Performance under different error bounds. To show that Auncel consistently outperforms the baselines when the error bound changes, we vary the error bound from 1% to 10% and run the top-100 workload on the SIFT10M dataset. Figure 11 shows the performance under different error bounds. Auncel achieves 1.4–1.7× lower average end-to-end latency than LAET and 1.8–4.8× lower of that than Faiss.

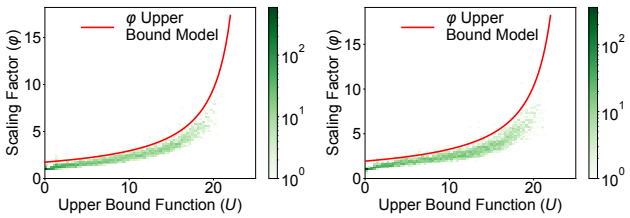
Performance under different values of k. We also vary the value of k from 10 to 100. We fix the error bound as 10% and run different top-k workloads on the SIFT10M dataset. From Figure 12, we observe that Auncel outperforms LAET and Faiss by 1.6–3.3× and 4.6–10× on average query latency, respectively. It confirms that Auncel can handle different values of k in top-k.

6.2 Effectiveness of ELP Techniques

In this set of experiments, we evaluate the effectiveness of the ELP building techniques in Auncel.

Error Profiles. To evaluate our error profiling technique, we run Auncel with the top-100 workload on the four datasets. We vary the error bound from 10% to 70%. Figure 13 illustrates the maximum and 95%-tile actual errors. We also plot the ideal straight lines (i.e., actual error equals to error bound) in Figure 13. Note that the measured actual error is always no bigger than the error bound, which demonstrates the ability of Auncel to guarantee bounded errors. As we increase the error bound, the measured actual error increases as well, indicating that Auncel adapts to the error bound. However, the maximum error does not increase monotonically with the error bound on SIFT10M, DEEP10M and GIST1M. This is because the geometric characteristics (e.g., dimension and distribution) of query vectors vary widely across different datasets.

Time Profiles. To evaluate our time profiling technique, we run Auncel with the top-100 workload on the four datasets and



(a) SIFT10M. (b) DEEP10M.

Figure 15: Validation of Formula 10.

report query latency of each query, with a time bound from 5 ms to 50 ms. Figure 14 shows the results of the maximum and minimum end-to-end query latencies. The results show that each query is terminated before the time bound.

6.3 Validation of Mathematical Formulation

This experiment validates that our theoretical model (Formula 10) fits well with real-world unstructured datasets. In ELP initialization, we compute the geometric upper bound function U and collect the corresponding real scaling factor φ according to the ground truth. We sample a portion of $U-\varphi$ pairs, where φ is the maximal in a small interval, to model the tight upper bound of φ. The larger interval apparently leads to a tighter upper bound. We then use least squares to fit Formula 10 for these $U-\varphi$ pairs. Figure 15 shows the results of the top-100 search workload on SIFT10M and DEEP10M. The results confirm that Formula 10 can well capture the relationship between U and φ, which allows Auncel to accurately predict the runtime errors.

6.4 System Overhead

Runtime overhead. To evaluate the runtime overhead of Auncel, we perform an experiment with top-k search workloads on different datasets. We configure Auncel to search fixed number of clusters and make an error prediction after searching each cluster. For comparison, we measure the ELP prediction time and the entire time of query processing. We also vary the value of k from 10 to 100, and run Auncel on SIFT10M. As shown in Table 3 and Table 4, the average latency can hardly be distinguished between Auncel with ELP and Auncel without ELP. The runtime overhead of using ELP in Auncel is within 1%. Note that the average latency almost stays the same from top-10 search to top-100 search. This is because distance computation dominates in the search process such that top-k relevant computation is negligible.

ELP Building Time. We evaluate the offline time taken for ELP building. We configure Auncel to train 50% query vectors with the top-100 workload on different datasets. Table 5 shows that the time to build ELP is within ten minutes, which is relatively small for datasets with ten million vectors.

6.5 Scalability

Auncel shards a dataset into several partitions. For a vector query, it performs local ANN search with ELP in the `map` phase and aggregates the local results in the `reduce` phase. We evaluate how configurations with different number of workers impact the average latency. We use four c5.metal EC2 instances to verify the scalability of Auncel. In this experiment, we adopt the calibration techniques described in §4.5 to guarantee 10% global bounded error for all queries. We vary the number of workers from 4 to 128, and run the top-100 workload on DEEP1B. Auncel assigns each machine an equal number of workers and spawns a leader worker on one of the machines. The leader worker receives all local top-k results and completes the reduce task. Figure 16 shows that the average latency is almost halved when the number of workers doubles each time, indicating that Auncel scales out *near ideally*. This is because Auncel only transmits the indexes of local top-100 vectors and their corresponding distances to the query vector from the workers to the leader; it does not transmit the raw data of the high-dimensional vectors. The `reduce` phase to aggregate local results takes only a small portion of the total time. Therefore, Auncel can fully leverage the advantages of data parallelism to scale out.

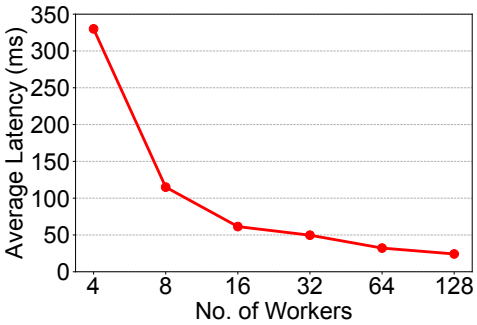


Figure 16: Scalability on multiple workers.

Dataset	Without ELP	With ELP
SIFT10M	68.02 ms	68.41 ms (+0.58%)
DEEP10M	48.16 ms	48.55 ms (+0.81%)
GIST1M	6.86 ms	6.91 ms (+0.70%)
TEXT10M	95.42 ms	96.00 ms (+0.61%)

Table 3: Runtime overhead under different datasets.

Top-K	Without ELP	With ELP
Top-10	68.09 ms	68.47 ms (+0.55%)
Top-50	68.12 ms	68.51 ms (+0.57%)
Top-100	68.02 ms	68.4 1ms (+0.58%)

Table 4: Runtime overhead under different values of k.

Dataset	Building Time
SIFT10M	6.20min
DEEP10M	4.44min
GIST1M	0.61min
TEXT10M	8.65min

Table 5: ELP building time on different datasets.

7 Discussion

Hardware acceleration. Some query engines [23, 49] leverage GPUs to accelerate vector query processing. GPU acceleration works well for small datasets, but is not suitable for large datasets because of limited GPU memory size. Thus, CPUs are widely used in production for large datasets. Also, GPUs are more expensive than CPUs. Therefore, we focus on CPUs for implementation and evaluation in this paper. We remark that the design of Auncel is orthogonal to the underlying hardware. Auncel can be applied to GPUs or other specialized hardware for vector query processing.

Vector compression. Vector compression [50] is proposed to reduce the memory footprint of large datasets. It compresses high-dimensional vectors into low-dimensional space while maintaining the distance property. The number of dimensions in each vector is reduced after vector compression. This technique is orthogonal to Auncel since we only consider the distance property between different vectors. Auncel processes vector queries on a dataset, no matter how many dimensions each vector in the dataset has.

8 Related Work

ANN indexes. Many ANN indexes are proposed to improve query accuracy and reduce query latency, such as IVF [15, 25, 35], graph index [16, 51, 52] and locality sensitive hashing [18, 53–55]. These algorithms perform search on sampling data which trade accuracy for query latency. They leave the sampling size (e.g., top-n in IVF) to users and do not provide bounded performance. These ANN algorithms are orthogonal and complementary to Auncel, and we follow one of the state-of-the-art solutions, IVF, to build Auncel. Besides, ANN algorithms also have different system characteristics [56]. For example, the graph index is more efficient than IVF, but it needs extra memory to hold large graphs. An interesting direction for future work is to build a unified ELP for more ANN indexes and provide bounded performance according to user preferences. Some recent works [30, 57] focus on early stopping conditions of nearest neighbor search to reduce average query latency at a high accuracy, but they do not provide any error or time bound guarantees. With the proliferation of unstructured data and machine learning, ANN on the embedding vectors of unstructured data becomes a key component in many AI applications, such as recommendation [1–4], recognition [5–8] and information retrieval [9–11]. Recent industrial vector data management systems [23, 24, 58] are developed to meet the rapidly increasing demand of these AI applications. They typically build their query processing engines on top of Faiss [22]. As we integrate the Auncel prototype into Faiss, it is convenient for these systems to leverage Auncel to improve vector query processing.

Approximate query processing. Approximate query processing systems [19, 20, 59–61] have gained a lot of popularity due to the long latency of exact search. These systems all provide time or latency guarantees through probability statistics. However, none of them pays attention to unstructured data represented by vectors. BlinkDB [19] and Quickr [60] focus on structured data and approximate aggregation jobs while ASAP [20] focuses on approximate graph pattern mining. The probability statistics method fails to produce good results on vector queries with performance guarantees. Thus, we introduce a novel high-dimensional geometry theory tailored for vector queries in Auncel. GRASS [21] is a scheduler for approximation jobs in data analytics clusters to alleviate the straggler problem in a map-reduce framework. It is complementary to the distributed design of Auncel since our error and latency calibration mechanism is easy to be integrated into such cluster schedulers. Big data warehouses [62–64] are prevalent in modern cloud services, which provide high-performance query processing. To manage large-scale unstructured data, Auncel can be integrated into these query systems to provide bounded performance. Auncel bridges the gap between approximate query processing and vector queries.

9 Conclusion

We present Auncel, a vector query processing engine that provides bounded errors and latencies on very large unstructured datasets. Auncel exploits the geometric properties of high-dimensional space and the nature of vector query processing to build precise and lightweight ELPs. Auncel is a distributed solution that leverages probability theory to scale out with multiple workers. We demonstrate the performance of Auncel on a variety of datasets. Auncel significantly reduces query latency while meeting error or latency bounds, and scales to billion-scale datasets with latency reduction.

Acknowledgments. We sincerely thank our shepherd Anurag Khandelwal and the anonymous reviewers for their valuable feedback on this paper. This work is supported by the National Key Research and Development Program of China under the grant number 2021YFB3300700, the National Natural Science Foundation of China under the grant number 62172008, the National Natural Science Fund for the Excellent Young Scientists Fund Program (Overseas), and a research gift from Moqi. Xin Jin and Xuanzhe Liu are the corresponding authors. Zili Zhang, Chao Jin, Xuanzhe Liu and Xin Jin are also with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education.

References

[1] A. S. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *WWW*, 2007.

[2] J. Suchal and P. Návrat, "Full text search engine as scalable k-nearest neighbor recommendation system," in

IFIP International Conference on Artificial Intelligence in Theory and Practice, 2010.

[3] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Recsys*, 2016.

[4] M. Grbovic and H. Cheng, "Real-time personalization using embeddings for search ranking at airbnb," in *ACM SIGKDD*, 2018.

[5] R. He, Y. Cai, T. Tan, and L. Davis, "Learning predictable binary codes for face indexing," *Pattern recognition*, 2015.

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[7] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[8] X. Liu, W. Liu, H. Ma, and H. Fu, "Large-scale vehicle re-identification in urban surveillance videos," in *ICME*, 2016.

[9] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, "The rise of deep learning in drug discovery," *Drug discovery today*, 2018.

[10] A. C. Mater and M. L. Coote, "Deep learning in chemistry," *Journal of chemical information and modeling*, 2019.

[11] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy, "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing," *Nature biotechnology*, 2015.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, 2015.

[13] R. Blumberg and S. Atre, "The problem with unstructured data," *Dm Review*, 2003.

[14] "Eighty Percent of Your Data Will Be Unstructured in Five Years." https://solutionsreview.com/data-management/.

[15] A. Babenko and V. Lempitsky, "The inverted multi-index," *TPAMI*, 2014.

[16] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *TPAMI*, 2018.

[17] K. Zhou, Q. Hou, R. Wang, and B. Guo, "Real-time kd-tree construction on graphics hardware," *ACM TOG*, 2008.

[18] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*, 2004.

[19] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "BlinkDB: queries with bounded errors and bounded response times on very large data," in *EuroSys*, 2013.

[20] A. P. Iyer, Z. Liu, X. Jin, S. Venkataraman, V. Braverman, and I. Stoica, "ASAP: Fast, approximate graph pattern mining at scale," in *USENIX OSDI*, 2018.

[21] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu, "GRASS: Trimming stragglers in approximation analytics," in *USENIX NSDI*, 2014.

[22] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, 2019.

[23] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, *et al.*, "Milvus: A purpose-built vector data management system," in *ACM SIGMOD*, 2021.

[24] C. Wei, B. Wu, S. Wang, R. Lou, C. Zhan, F. Li, and Y. Cai, "Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data," in *Proceedings of the VLDB Endowment*, 2020.

[25] Q. Chen, B. Zhao, H. Wang, M. Li, C. Liu, Z. Li, M. Yang, and J. Wang, "SPANN: Highly-efficient billion-scale approximate nearest neighbor search," in *Neural Information Processing Systems*, 2021.

[26] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal lsh for angular distance," in *Neural Information Processing Systems*, 2015.

[27] M. Muja and D. Lowe, "Flann-fast library for approximate nearest neighbors user manual," *VISAPP*, 2009.

[28] "Annoy." https://github.com/spotify/annoy.

[29] L. Boytsov and B. Naidan, "Engineering efficient and effective non-metric space library," in *SISAP*, 2013.

[30] C. Li, M. Zhang, D. G. Andersen, and Y. He, "Improving approximate nearest neighbor search through learned adaptive early termination," in *ACM SIGMOD*, 2020.

[31] "yandex billion-scale datasets." https://research.yandex.com/datasets/biganns.

[32] M. Grohe, "word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data," in *ACM SIGMOD*, 2020.

[33] H. Chen and H. Koga, "Gl2vec: Graph embedding enriched by line graphs with edge features," in *Neural Information Processing Systems*, 2019.

[34] G. C. Tomas Mikolov, Kai Chen and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations (ICLR)*, 2013.

[35] D. Baranchuk, A. Babenko, and Y. Malkov, "Revisiting the inverted indices for billion-scale approximate nearest neighbors," in *ECCV*, 2018.

[36] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, 1979.

[37] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Neural Information Processing Systems*, 2017.

[38] "Intel oneAPI Math Kernel Library (oneMKL)." https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/api-based-programming/intel-oneapi-math-kernel-library-onemkl.html.

[39] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, 2008.

[40] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *USENIX HotCloud Workshop*, 2010.

[41] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: re-rank with source coding," in *ICASSP*, 2011.

[42] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *TPAMI*, 2010.

[43] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," in *SISAP*, 2017.

[44] S. Harsha, W. George, A. Martin, B. Artem, B. Dmitry, C. Qi, D. Matthijs, K. Ravishankar, S. Gopal, S. Suhas, and W. Jingdong, "Billion-scale approximate nearest neighbor search challenge," in *Neural Information Processing Systems Competition Track*, 2021.

[45] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *IJCV*, 2001.

[46] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[47] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *CIKM*, 2013.

[48] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[49] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale commodity embedding for e-commerce recommendation in alibaba," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 839–848, 2018.

[50] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[51] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," in *Proceedings of the VLDB Endowment*, 2019.

[52] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnawamy, and R. Kadekodi, "Diskann: Fast accurate billion-point nearest neighbor search on a single node," 2019.

[53] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *Proceedings of the VLDB Endowment*, 2007.

[54] Y. Zheng, Q. Guo, A. K. Tung, and S. Wu, "Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index," in *ACM SIGMOD*, 2016.

[55] L. Gong, H. Wang, M. Ogihara, and J. Xu, "idec: indexable distance estimating codes for approximate nearest neighbor search," in *Proceedings of the VLDB Endowment*, 2020.

[56] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," *Information Systems*, 2020.

[57] A. Gogolou, T. Tsandilas, T. Palpanas, and A. Bezerianos, "Progressive similarity search on time series data," in *BigVis*, 2019.

[58] W. Yang, T. Li, G. Fang, and H. Wei, "Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension," in *ACM SIGMOD*, 2020.

[59] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, "Approxhadoop: Bringing approximations to mapreduce frameworks," in *ACM ASPLOS*, 2015.

[60] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding, "Quickr: Lazily approximating complex adhoc queries in bigdata clusters," in *ACM SIGMOD*, 2016.

[61] A. P. Iyer, A. Panda, S. Venkataraman, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica, "Bridging the gap: towards approximate graph analytics," in *SIGMOD GRADES-NDA*, 2018.

[62] "Google BigQuery." https://cloud.google.com/bigquery/.

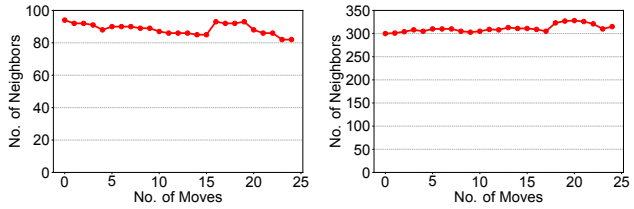
[63] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, *et al.*, "The snowflake elastic data warehouse," in *Proceedings of the 2016 International Conference on Management of Data*, 2016.

[64] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan, "Amazon redshift and the case for simpler data warehouses," in *ACM SIGMOD*, 2015.

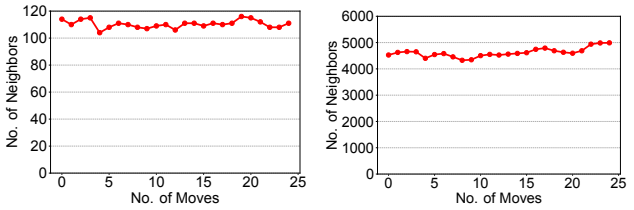
[65] "CMU CStheory-infoage chap1-high-dim-space." https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/chap1-high-dim-space.pdf.

A Appendix

A.1 Validation of Local Uniform Distribution



(a) cp_1 in SIFT10M. (b) cp_2 in SIFT10M.



(c) cp_3 in DEEP10M. (d) cp_4 in DEEP10M.

Figure 17: Validation of local uniform distribution.

We emphasize that the assumption about data distribution is *local* uniform distribution rather than uniform distribution. It is because the real-world items are distributed together smoothly. We perform a measurement to validate that vectors conform to local uniform distribution in widely-used unstructured datasets. We randomly pick a central point (cp) in the geometric space and let it do a random walk. Each time cp moves a small distance in either direction, the number of database vectors within a small radius (r) from cp is counted through a sweep. In this measurement, we pick two initial central points for SIFT10M and DEEP10M, respectively. Figure 17 shows the dynamics of the number of cp's neighbors and how it changes when cp moves. Within 25 moves (i.e., a relatively small local scope), the number of cp's neighbors fluctuates no more than 14% in the four examples. Consequently, it is reasonable to conclude the local uniform distribution.

A.2 Analysis of Formula 10

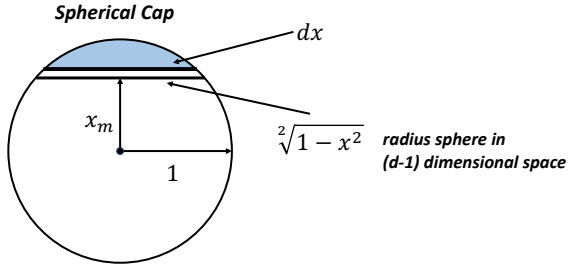


Figure 18: Unit spherical cap.

Let db_m represent the distance between the query vector q and the boundary of C_1, C_m. We leverage cosine theorem to calculate db_m by three anchor vectors, q and the centroid vectors of C_1, C_m, where the boundary is the mid-vertical plane of the line segment connecting the two centroids.

Since only the volume ratio is considered, we simplify the model into unit sphere, which substitutes $\lambda_i(j)$ with unit element (1) and db_m with $x_m = \frac{db_m}{\lambda_i(j)}$ after processing C_i. Thus, $\frac{V(P_j(m))}{V(B(\lambda_i(j)))}$ is identical to the ratio of spherical cap's volume, $V(sc)$ to the volume of the unit sphere as Figure 18 shows. According to properties of high-dimensional space [65], the volume of the unit sphere, $Un(d)$ in d-dimensional space is

$$V(Un(d)) = \frac{\pi^{\frac{d}{2}}}{\frac{d}{2}\Gamma(\frac{d}{2})}.$$

As for the spherical cap, the exact volume calculation is time-consuming through definite integral. To guarantee the bounded error, it is sufficient to provide an upper bound formula of the ratio. Thus, we derive the following lemma.

Lemma A.1 *For any $0 \le x_m \le 1$, the fraction of the volume of the hemisphere above the boundary (spherical cap, sc) is $V(sc)$. The unit sphere's volume is $V(Un(d))$ in d-dimensional space ($d \ge 3$). We conclude that $\frac{V(sc)}{V(Un(d))} \le a \times \arccos(x_m)$, where a is a number corresponding to d.*

Proof. The surface area of the intersection of the boundary and the sphere is

$$(1-x^2)^{\frac{d-1}{2}} V(Un(d-1)).$$

Thus, the volume of sc is:

$$V(sc) = V(Un(d-1)) \int_{x_m}^{1} (1-x^2)^{\frac{d-1}{2}} dx.$$

Now,

$$\begin{aligned} \frac{V(sc)}{V(Un(d))} &= \frac{V(Un(d-1))}{V(Un(d))} \int_{x_m}^{1} (1-x^2)^{\frac{d-1}{2}} dx \\ &\le a \times \int_{x_m}^{1} (1-x^2)^{\frac{d-1}{2}} dx. \end{aligned}$$

As for the upper bound, since $0 \le (1-x^2) \le 1$ and $-\frac{1}{2} < 0 < \frac{d-1}{2}$, we have $(1-x^2)^{\frac{d-1}{2}} \le (1-x^2)^{-\frac{1}{2}}$. Due to the properties of definite integral,

$$\int_{x_m}^{1} (1-x^2)^{\frac{d-1}{2}} dx \le \int_{x_m}^{1} (1-x^2)^{-\frac{1}{2}} dx$$

Thus,

$$\frac{V(sc)}{V(Un(d))} \le a \times \int_{x_m}^{1} (1-x^2)^{-\frac{1}{2}} dx = a \times \arccos(x_m).$$

The lemma is concluded by the above proof, and we fit a by sampling queries offline.

Above all, one of U is $\sum_{m=i+1}^{l} \arccos(x_m)$ in Formula 10 where $a \times U$ represents the upper bound of $\frac{\sum_{m=i+1}^{l} V(P_j(m))}{V(B(\lambda_i(j)))}$.