2023 9th International Conference on Big Data and Information Analytics (BigDIA) | 979-8-3503-3007-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/BigDIA60676.2023.10429609

A Brief Survey of Vector Databases

1st Xingrui Xie
College of Systems Engineering
National University of Defense Technology
Changsha, China
xiexingrui20@nudt.edu.cn

2nd Han Liu
College of Systems Engineering
National University of Defense Technology
Changsha, China
liuhan20@nudt.edu.cn

3rd Wenzhe Hou
College of Systems Engineering
National University of Defense Technology
Changsha, China
wzhou@nudt.edu.cn

4th Hongbin Huang[†]
Laboratory for Big Data and Decision
National University of Defense Technology
Changsha, China
hbhuang@nudt.edu.cn

***Abstract*—The explosive growth of massive high-dimensional data requires capabilities for data processing, storing, and analyzing. This brings significant challenges to traditional databases due to the poor ability to handle high-dimensional data and its original design for stand-alone machines. Fortunately, vector databases have provided a practical solution for the management and analysis of high-dimensional data. Especially, they retrieve results related to the query efficiently after encoding various forms of data (e.g., text, image, and video) into vectors. The purpose of this paper is to offer insight into vector databases by presenting a brief survey. Firstly, the workflow of vector databases including indexing and querying, is detailed along with a specific case. Subsequently, we elaborate on the related methods applied in vector databases, which are the core techniques to enhance search efficiency and reduce computational overhead, particularly similarity search algorithms and similarity metrics. Further, we introduce widely used vector database products (e.g., Pinecone, Chroma, and Milvus) and compare them from multiple factors that should be taken into consideration. We also discuss potential avenues for future research in this domain. To conclude, this survey provides a comprehensive understanding of vector databases for retrieval from vast high-dimensional datasets.**

***Index Terms*—high-dimensional data, nearest neighbors' search, vector databases, similarity search, similarity metrics**

I. Introduction

In recent years, big data has deeply influenced our lives. Particularly, artificial intelligence (AI) is widely applied due to the natural user interface and its convenience. For example, OpenAI's ChatGPT [1] is capable of responding to various natural language texts, while NaturalSpeech 2 text-to-speech (TTS) system [2] and CycleGAN [3] show impressive ability of generating speech and images, respectively. A key to the success of AI is efficiently analyzing the vast data with various forms (e.g., text, audio, and image) [4], [5]. The urgent demand for data management prompts the advancement of database technology in multiple industries, such as education [6], healthcare [7], and fashion [8].

As mentioned above, managing and processing vast volumes of data in various forms is necessary for big data applications. However, traditional databases fail to process such data because of their inherent shortcomings [9], [10]. Specifically, four points are listed in the following:

- Data scale. With limited storage capacity and processing power, traditional databases are inadequate for handling large-scale data.
- Data structure. Traditional databases solely cater to structured data and are incapable of managing semi-structured or unstructured data.
- Scalability of data. The scalability of traditional databases is challenging when it comes to horizontal scaling, which involves adding more nodes to accommodate increasing data volumes.
- Access restrictions. The access restrictions of traditional databases limit their accessibility to only the SQL language, thereby lacking support for non-relational data.

Although some techniques can address these existing issues, they bring massive computational costs [11]. On the contrary, vector databases, whose basic data models are vectors, not only have the capability to deal with high-dimensional and unstructured data but also enable impressively efficient retrieval. Actually, vector databases are designed to store and manage vector embeddings, which represent features and properties of data in various forms.

The primary benefits of vector databases in managing substantial volumes of data are presented below [12]–[14]:

- Vectors as the fundamental structures. Vector databases are designed for vector-based storage and management, as well as transforming multi-modal data into vector representations. Thus they can quickly deal with high-dimensional vectors and unstructured data, including audio, text, and video.
- Ease to scale out. Because of the vectorized storage and access, vector databases can fully leverage computing resources through parallel processing, thereby enabling rapid retrieval.

Due to their special benefits, vector databases are crucial in various complex tasks according to current studies. For example, literature [15] has represented the structural information of the fuzzy resource description framework (RDF) graph in the vector space through the establishment of a model for embedding

[†] Corresponding author.

fuzzy RDF knowledge graphs. This certification confirms the transformation's effective support for various downstream tasks involving fuzzy data. Literature [16] investigated the Behavior2vector, a method representing users' personalized travel behaviors as vectors, offering a more rational and efficient approach compared to the existing graph embedding techniques tested on travel big data. Literature [17] extended the common x-vector and a Bayesian formulation for speaker embedding via coalescing uncertainty modeling, which can drastically reduce error rates and detection costs.

Recently, multiple vector databases have been accepted and widely used. For instance, Faiss[1], developed by Facebook, is an open-source engine for high-efficiency similarity retrieval and clustering of dense vectors. It performs similarity vector queries using methods like Euclidean Distance and Dot Product. Additionally, pgvector[2] is designed to enhance PostgreSQL's capabilities by providing outstanding storage, querying, and processing of vector data. By seamlessly integrating with PostgreSQL, it empowers users to leverage their SQL knowledge for performing vector queries. Furthermore, the current popular open-source vector databases, Chroma[3] and Milvus[4], provide support for Large Language Model (LLM) and machine-learning tasks. Despite being a closed-source solution, Pinecone[5] demonstrates excellent capabilities in processing billions of vector data points swiftly and updating the index in real time.

There are now a variety of techniques that have been developed around vector databases. Various vector database products have also emerged. There is, however, a dearth of systematic research on the theoretical underpinnings and performance of vector database products. Therefore, the paper provides a brief survey of vector databases to addresses the above questions and makes following contributions:

- We hierarchically induce the basic workflow of vector database, and further exemplify it with a real case.
- We detail the technical aspects of vector databases, including similarity search algorithms and similarity metrics.
- We engage in a beneficial comparison of popular vector database products in various perspectives and provide valuable insights for future research.

The remainder of the paper is structured as follows. Section II provides an explanation of the workflow. Section III expatiates the related methods of vector databases, including corresponding similarity search algorithms and similarity metrics. Section IV discusses the essential criteria for selecting a vector database and showcases the mainstream options available. Section V suggests the trends and potential application in this domain. Finally, Section VI concludes the paper.

II. Workflow of Vector Databases

The workflow of vector databases is the foundation for database operation, consisting of indexing and querying. The section commences with an overview of the vector databases' general process. Thereafter we present a comprehensive illustration of indexing and querying through a specific case.

A. *Overview of Vector Databases' Workflow*

The overall workflow of vector databases usually consists of two phases: indexing and querying, as shown in Fig. 1.

Firstly, indexing encompasses transformation and compression. Via **transformation**, various complex data are converted into vector embeddings and represent desired information. Additionally, through employing **compression**, a multitude of vector embeddings can be classified based on the compression algorithms, and the class should be represented by a vector referred to as *representative vector*. The higher the similarity of the vectors, the greater the probability of them being stored in the same class. The "class" exists as a data structure, with its type being determined by the compression algorithms.

Besides, querying covers transformation, rough-comparison, detailed-comparison, and retrieval. The **transformation** here is equivalent to the transformation used for indexing. Then during **rough-comparison**, the query is compared with the *representative vector* stored by each class in the vector database, aiming to identify the class that exhibits the highest similarity to the query. In the following step, **detailed-comparison**, the query is compared with each vector data in its most similar class one by one, resulting in the identification of the vector data that exhibits the highest similarity to the query. The corresponding realistic representation of this identified vector data is then **retrieved** as the final result.

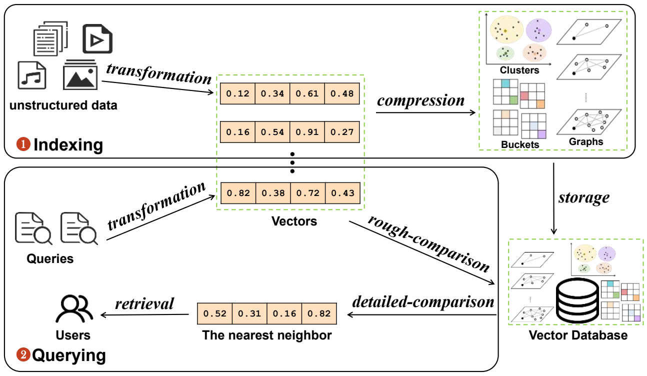


Fig. 1. Workflow of Vector Databases.

B. *A Running Example for Indexing*

To gain a more comprehensive understanding of the fundamental workflow of vector databases, we delve deeper into a running example. For example, we store image data, n types of dogs, in the vector database V. We distinguish between different types of dogs according to their body and hair. Set D_i to denote the i-th type of dog, where $i = 1, 2, ..., n$, illustrated in Fig. 2, here $n = 8$.

First, in **transformation**, map each dog as a corresponding two-dimensional vector, achieved by quantifying its characteristics on a closed interval ranging from 0 to 1. The vector for the i-th type of dog is represented by $v_i = (body_i, hair_i)$.

Then, in **compression**, to avoid the excessive variety of dogs resulting in significant storage space occupation, utilize a compression algorithm for dog classification. Take the K-Means compression algorithm as an example. Calculate the Euclidean distance between v_i and v_j, denoted as $d_{ij} (i = 1, 2, ...n, j = 1, 2, ..., n, i \neq j)$ and set the distance threshold $distance$. When

[1] https://faiss.ai/
[2] https://pgxn.org/dist/vector/0.4.4/
[3] https://docs.trychroma.com/
[4] https://milvus.io/
[5] https://www.pinecone.io/

$d_{ij} \leq distance$, v_i and v_j are grouped into a cluster instead of different clusters. The average value of each cluster is computed after forming umpty clusters in this manner, resulting in a vector object known as the cluster centroid that serves as the *representative vector* .

Herein, record annotations for future reference: use N to represent the number of clusters, use c_k to represent the k-th cluster, use C_k to represent the k-th cluster centroid, then $k = 1, 2, ..., N$.

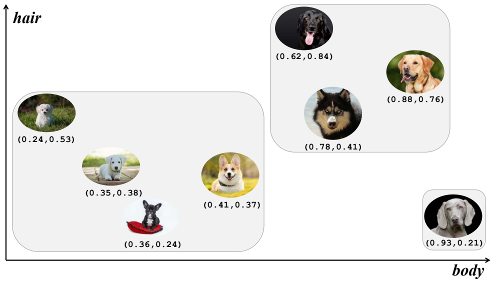


Fig. 2. Schematic diagram of Indexing (K-Means).

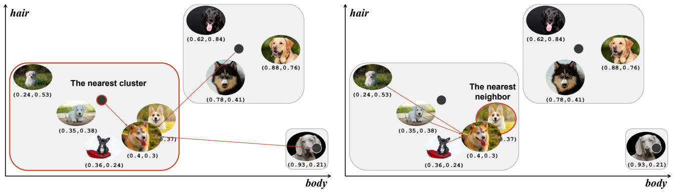
Fig. 2 shows that maltese, puppy, bulldog, and corgi belong to the same cluster; hovawart, husky, and labrador are grouped together in another cluster; whereas weimaraner stands alone in its own cluster. Therefore, the real-life dogs are successfully transformed into vector form and stored in the vector database during the indexing phase.

C. A Running Example for Querying

Benefited from the vector database V built in the indexing phase previously, we can take a closer look at the querying phase. For example, we enter the dog image q and query the dog most similar to it from V.

First, in **transformation**, q maps to the vector form v_q in the same way as the transform operation in indexing. Suppose $v_q = (0.4, 0.3)$.

Then, the Euclidean distance is taken as an example to calculate the similarity. In **rough-comparison**, the similarity between N cluster centers in V and v_q are respectively compared, and the r-th cluster center with the highest similarity to v_q is assumed to be C_r. And then in **detailed-comparison**, compare v_q with the vector objects in the r-th cluster one by one and finally get the vector object v_{qr} with the highest similarity to v_q. Hence, v_{qr} is the result of the query q, and its corresponding dog image is **retrieved**. The above comparison process is shown in Fig. 3.



(a) Rough-comparison.

(b) Detailed-comparison.

Fig. 3. Schematic diagram of querying (Euclidean distance).

In this running case, based on the input image of the dog, we referred to our vector database and identified the most closely matched dog, which happened to be a corgi.

III. Methodological Support for Vector Databases

To optimize data storage and enable rapid querying, vector databases employ specialized methods. This section provides a deep analysis and comparison of the relevant approaches, focusing on two perspectives: similarity search algorithms and similarity metrics.

A. Similarity Search Algorithms

Vector databases emphasize similarity search and employ various algorithms to facilitate rapid querying. These algorithms can compress the original vectors and provide a ranked list of vectors with the highest similarity score to the query vectors. This part explores prevalent algorithms like K-Means, Locality Sensitive Hashing, Hierarchical Navigable Small Worlds, and Product Quantization including their methodologies for addressing vector embeddings.

1) K-Means: A clustering technique excels at partitioning vectors in vector databases into distinct categories. During querying, we first need to estimate the cluster to which the search vector belongs. Subsequently, search within this specific cluster. Thus, the search scope is significantly reduced. The flow of K-Means is shown in Fig. 4.

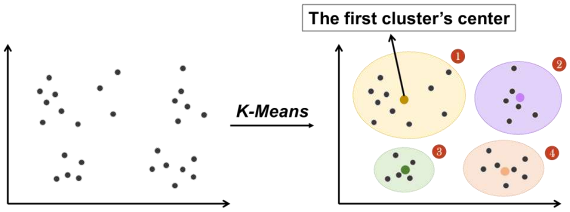


Fig. 4. Schematic diagram of K-Means ($k = 4$).

K-Means focuses on unsupervised learning by considering the distance between data objects, like the Euclidean Distance, as the standard for measuring similarity. The closer the proximity between data objects, the stronger their resemblance and the more likely they are to be grouped together in a cluster. Furthermore, the values of the k cluster centers are computed as the mean of the values within their respective clusters. The following pseudo-code, Alg. 1, outlines the fundamental steps of this process [18].

2) Locality Sensitive Hashing: It is also designed for rapid search and analysis of high-dimensional data. The primary contribution of Locality Sensitive Hashing (LSH) is to map similar elements into the same hash buckets, thereby reducing the number of comparisons required instead of relying on an exhaustive search [19].

The primary goal in dictionary lookup is to minimize occurrences of hash collisions, which occur when multiple key values are mapped to the same bucket. The design of LSH functions is specifically aimed at maximizing collisions, albeit only for input values that are closer together. LSH relies on the utilization of locality-sensitive hash functions, the formal definition of which is presented below [20], [21]:

Set a family of hash functions as $\mathcal{H} = \left\{h : \mathbb{R}^d \rightarrow U\right\}$, mapping $\mathbb{R}^d$ to some universe U. For any two points p and q in d-dimensional space, they are deemed to collide over h, if they are mapped to the same bucket, also called hash table, through functions in functions family $\mathcal{H}$, namely $h(p) = h(q)$.

The family $\mathcal{H}$ is called (r, cr, p_1, p_2)-locality-sensitive, if it satisfies the following conditions for $\forall p, q \subseteq \mathbb{R}^d$, where r is the cut-off distance for nearness $r \geq 0$, is an approximation ratio

Algorithm 1: K-Means

```
   Input  : Sample set $D = \{x_1, x_2, ..., x_N\}$, number of
            classification $k = 2$, maximun iteration number
            $M$, two random points $\{u_1, u_2\}$ from the
            subsample as the cluster centers.
   Output: Sample cluster centroids $\{C_1, C_2\}$
 1 for $m = 1 \to M$ do
 2     $C_1 \Leftarrow \emptyset, C_2 \Leftarrow \emptyset$; for $i = 1, 2, \ldots, N$ do
 3         $d_{i1} \Leftarrow \|x_i - u_1\|^2, d_{i2} \Leftarrow \|x_i - u_2\|^2$;
 4         if $d_{i1} \leq d_{i2}$ then
 5             $C_1 \Leftarrow C_1 \cup \{x_i\}$
 6         else
 7             $C_2 \Leftarrow C_2 \cup \{x_i\}$
 8         end
 9     end
10     $\tilde{u_1} \Leftarrow \frac{1}{|C_1|} \sum_{x \in C_1} x, \tilde{u_2} \Leftarrow \frac{1}{|C_2|} \sum_{x \in C_2} x$;
11     if $(\tilde{u_1} == u_1)$ & & $(\tilde{u_2} == u_2)$ then
12         break from line 3
13     else
14         $u_1 \Leftarrow \tilde{u_1}, u_2 \Leftarrow \tilde{u_2}$
15     end
16 end
17 return $C_1, C_2$
```

$c > 1$, and p_1, p_2 are collision probabilities, $p_1, p_2 \in [0, 1], p_1 > p_2$:

- If $||p - q|| \leq r$ then $Pr[h(p) = h(q)] \geq p_1$;
- If $||p - q|| > cr$ then $Pr[h(p) = h(q)] \leq p_2$,

where $h \in \mathcal{H}$ chosen randomly. In high-dimensional space, LSH firstly indexes massive amounts of data and then searches, as shown in Fig. 5.

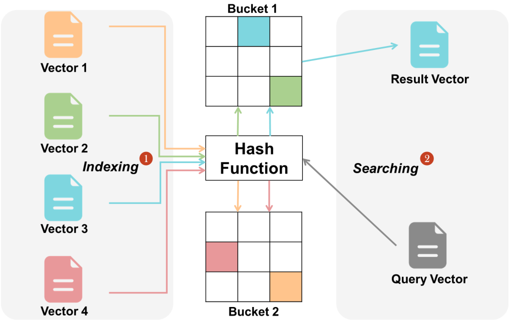


Fig. 5. Process of Locality Sensitive Hashing.

- **Indexing**
  Select LSH hash functions satisfied with (r, cr, p_1, p_2)-locality-sensitive; determine the number of hash tables and functions in each hash table as well as other relevant parameters of the functions; put data into buckets based on hashing facilitating the construction of hash tables.
- **Searching**
  Acquire the index of the bucket for query by hashing; extract data from the corresponding table; calculate and analyze the similarity between query and data and retrieve the nearest neighbors.

3) Hierarchical Navigable Small Worlds: Furthermore, Hierarchical Navigable Small World (HNSW) is highly regarded for its fast search speeds and remarkable recall making it one of the top-performing indexes for vector similarity search [22].

HNSW establishes a hierarchical and fully graph-based structure, wherein the vertices represent vectors and the edges between them indicate the similarity between relevant vectors. The essence of HNSW is segregating inter-layer connections based on their lengths, enabling rapid search within a multi-layer graph [23], [24]. The graph structure and search procedure of HNSW are depicted in Fig. 6.

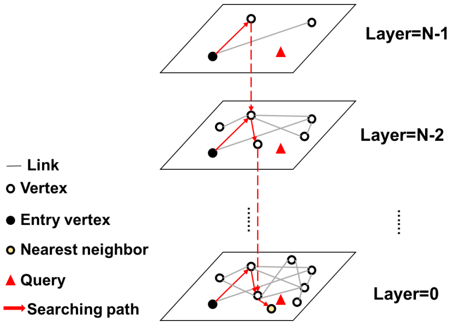


Fig. 6. Schematic diagram of Hierarchical Navigable Small World's graph structure and search procedure.

For graph structure, HNSW introduces the probability skip list by building several layers of linked lists. Each layer maintains vertices and links. Additionally, the number and length of skips/links decrease gradually as they move from the top layers to the lower layers. In such a structure, the coarse-grained top layers are appropriate for fast searches, whereas the finest-grained lower layers are qualified for accurate searches. Thus, HNSW can significantly enhance the speediness and accuracy of searching in high-dimensional spaces.

Besides, the search procedure always begins at a pre-defined entry vertex lying on the topmost layer. When searching, HNSW greedily traverses in upper layers by identifying the nearest neighbor vertices, until hitting a local minimum. Then the search shifts to the lower layer which has shorter links. Subsequently, repeat the search process until finding the local minimum of the bottom layer, acting as the nearest neighbor.

4) Product Quantization: Apart from the above algorithms, Product Quantization (PQ) is widely acknowledged as a compression method that delivers cutting-edge performance in exponentially growing data search problems. In other words, the memory footprint of indexes and computational costs can be reduced by PQ's compact code [25].

The superiority of PQ is its core principle of decomposing the high-dimensional space into a Cartesian product of individually quantized low-dimensional subspaces [26]. Take it as the instruction, there is a detailed process of PQ illustrated in Fig. 7 taking "$m = 3, k^* = 6$" as an example, where m denotes the number of segmentations for high-dimensional vectors and k^* represents the number of centroids in codebooks for low-dimensional vectors.

The vector quantizers lay the foundations for PQ utilizing compact codes. Assume that a vector quantizer is a function q that maps a d-dimensional vector x to a vector c_i embraced in a set of vectors C, where c_i and C are known as centroids and codebook with the size of k, respectively:

$$q : \mathbb{R}^d \longrightarrow C = \{c_0, c_1, c_2, ..., c_{k-1}\}, \quad (1)$$

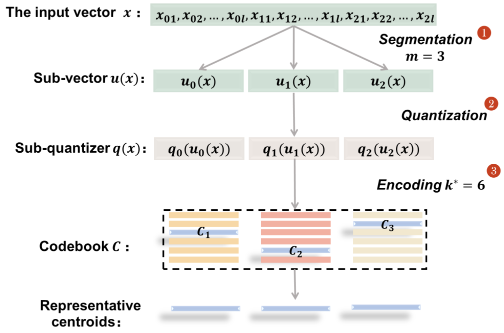


Fig. 7. Process of Product Quantization.

$$q(x) = C\left[i\right] = c_i. \tag{2}$$

And the quantization task is to achieve the nearest centroid:

$$q(x) = \arg\min_{c_i \in C} \|x - c_i\|. \tag{3}$$

The given vector x of dimensionality d is representatively divided into m distance sub-vectors $u_j(x)$, $0 \leq j \leq m-1$, of dimensionality $d^* = d/m$, where is a multiple of m. Usually, it is wise to make a trade-off between the accuracy of the representation and the computational cost of searching for m. And then, m distance sub-quantizers q_j, $0 \leq j \leq m-1$, quantize separately sub-vectors $u_j(x)$. Aforementioned segmentation and sub-quantization can be represented:

$$\begin{aligned} & x = \left(x_0, \ldots, x_{d^*-1}, \ldots, x_{d-1}\right), \\ & \rightarrow u\left(x\right) = \left(u_0(x), \ldots, u_{m-1}(x)\right), \\ & \rightarrow q\left(x\right) = \left(q_n\left(u_n\left(x\right)\right), \ldots, q_{m-1}\left(u_{m-1}\left(x\right)\right)\right). \end{aligned} \tag{4}$$

Let I, the product index set, and C, the codebook, literally correlate with the set of sub-quantizer $q(x)$. The product index set is defined as the Cartesian product $I = I_0 \times I_0 \times \ldots \times I_{m-1}$, while the codebook is expressed as $C = C_0 \times C_0 \times \ldots \times C_{m-1}$. The number of reproduction values in each sub-quantizer q_j is equal to that of centroids in each sub-codebook C_j, denoted as a constant value k^*. On this occasion, the total size of centroids is given by $k = (k^*)^m$. Therefore, it can be seen that PQ is competent to generate numerous centroids from sub-quantizers with small-size centroids k^* to handle large dimensionality.

5) Comparison among the similarity search algorithms: Confronted with vast amounts of data, the similarity search algorithms employed in vector databases primarily concentrate on enhancing search efficiency from two aspects: shrinking search space and reducing vector quantity. The former narrows down the search to only the closest clusters by employing clustering or organizing vectors into a tree-based or graphical structure, while the latter reduces the dimensionality of vectors, resulting in a shorter representation length. The optimization strategies and common applicable scenarios of the mentioned algorithms are summarized in Table I.

In general, K-Means partition the data point into k clusters, ensuring a high level of intra-cluster similarity and a low level of inter-cluster similarity. LSH effectively groups similar data points into designated buckets, thereby significantly improving the efficiency of similarity search. HNSW incorporates jump and local search techniques through the construction of a hierarchical graph structure, enabling linkage between similar data points within the graph. PQ focuses on the compression and similarity search of high-dimensional vectors.

TABLE I
METHODS TO IMPROVE SEARCH EFFICIENCY AND APPLICATION CASES OF SIMILARITY SEARCH ALGORITHMS

The Way to Optimize Search	Similarity Search Algorithms	High Dimension	Application Cases
Search space	K-Means	No	Image Segmentation
	LSH	Yes	Matching Text
	HNSW	Yes	Speech Recognition
Vector quantity	PQ	Yes	Image Retrieval, Audio Retrieval

Therefore, K-Means is well-suited for clustering continuous data, whereas PQ and LSH are applicable to similarity search tasks involving high-dimensional data. In addition, HNSW can be employed for similarity search in high-dimensional data and has the capability to construct a hierarchical structure.

B. Similarity Metrics

Particularly, vector databases also apply similarity metrics to compute the proximity between two vectors, thereby retrieving results, like Euclidean distance, Dot product similarity, and Cosine similarity. This part provides a deep insight into the theoretical foundations and practical applications of them.

1) Euclidean Distance: It is a fundamental instrument for measuring distances representing the straight-line distance between two points in hyperspace, widely used in various applications such as Node Vector Distance [27] and text document categorization [28].

Consider a set of points $\left\{r_i \in \mathbb{R}^2 | i = 1, 2, \ldots, N\right\}$ in two dimensions. The coordinates of r_i are (x_{1i}, x_{2i}). Here is the equation for calculating the Euclidean Distance between two points r_j and r_k and Fig. 8(a) visualizes its meaning:

$$\begin{aligned} d\left(r_j, r_k\right) = \sqrt{\left(x_{1j} - x_{1k}\right)^2 + \left(x_{2j} - x_{2k}\right)^2}, \\ j, k \in \left\{1, \ldots, N\right\}. \end{aligned} \tag{5}$$

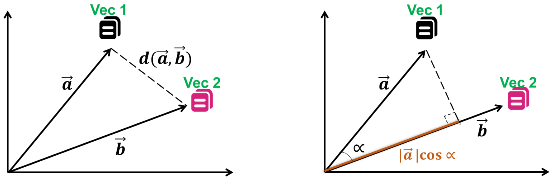
Then, $\left\{r_i \in \mathbb{R}^n | i = 1, 2, \ldots, N\right\}$ can depict the n-dimension locations, according to Euclidean Distance:

$$\begin{aligned} d\left(r_j, r_k\right) = \sqrt{\sum_{p=1}^{n} (x_{pj} - x_{pk})^2}, \\ j, k \in \left\{1, \ldots, N\right\}. \end{aligned} \tag{6}$$

Euclidean Distance, in particular, serves as a visualized similarity metric in space that directly reflects the magnitude of separation between each pair of compared vectors. The lower the value, the greater the similarity between vectors. Set and $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$ illustrating n different properties respectively, Eq. 7 expresses the formula for the Euclidean Distance between them:

$$d\left(\vec{a}, \vec{b}\right) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}. \tag{7}$$

Euclidean Distance is commonly regarded as an indicator for large vector databases due to its availability in calibration distances. For instance, literature [29] conducted a novel way called i-vectors scoring based on the Algerian Modern Colloquial Arabic Speech Corpus to match the weighted Euclidean Distance, which can effectively support real-life speaker verification scenarios.



(a) Euclidean Distance.

(b) Dot Product.

Fig. 8. Schematic diagram of similarity metrics in a two-dimensional space.

2) Dot Product Similarity: Additionally, Dot Product is a convenient metric for quantifying the similarity between two vectors. Set vectors $\vec{a} = (a_1, a_2, ..., a_n)$ and $\vec{b} = (b_1, b_2, ..., b_n)$, and their Dot Product is calculated as follows:

$$d\left(\vec{a},\vec{b}\right) = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + ... + a_n b_n, \tag{8}$$

where $\vec{a}$ and $\vec{b}$ are vectors being studied. Actually, Dot Product is the addition of the product of each component.

Dot Product can also be expressed as the scalar product of the magnitudes of the vectors and the cosine of the angle, denoted α between them. The formula is presented in Eq. 9, illustrated in Fig. 8(b):

$$\vec{a} \cdot \vec{b} = |\vec{a}| \left|\vec{b}\right| \cos \alpha, \tag{9}$$

where $|\vec{a}|$ and $|\vec{b}|$ represent the magnitudes of the vectors $\vec{a}$ and $\vec{b}$, respectively.

The advent of machine learning and LLM has rendered the utilization of the Dot Product justifiable. For instance, literature [30] has developed an abstractive text summarizer using natural language processing and observed that the Dot Product superior summaries for long texts. The fusing of global and local characteristics in [31] has led to an effective dot-product attention mechanism, which greatly contributed to computer vision. Additionally, literature [32] introduced the dot-product engine to enhance data searching and knowledge extraction in handling massive similarity operations.

3) Cosine Similarity: It is another similarity metric for two vectors, used to measure the similarity between two n-dimensional vectors in an n-dimensional space, by taking the dot product of the vectors and dividing it by the product of their magnitudes.

Set vectors $\vec{a} = (a_1, a_2, ..., a_n)$ and $\vec{b} = (b_1, b_2, ..., b_n)$, the Cosine for them is calculated as follows:

$$sim(\vec{a},\vec{b}) = \cos\alpha = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \left|\vec{b}\right|}, \tag{10}$$

where α is the angle between the vectors. Further, Eq. 10 describes the dot product by multiplying vectors' components, while Eq. 11 sums them together and expresses the vectors' length by taking the square root of the sum of the squares of the components.

$$sim(\vec{a},\vec{b}) = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{j=1}^{n} a_j^2}\sqrt{\sum_{j=1}^{n} b_j^2}}. \tag{11}$$

Cosine similarity, as defined by the trigonometric function, spans from -1 to 1 ($sim(\vec{a}, \vec{b} \in [-1, 1])$). In particular, a value of 1 represents complete consistency, while a value of 0 signifies exact dissimilarity. A value of -1 indicates opposite directions within the specified range. Cosine similarity, compared to Euclidean Distance and Dot Product, places greater emphasis on the angular difference between two vectors rather than their magnitude. Hence, the similarity of the vectors is determined by the magnitude of the angle between them. Generally, a smaller angle indicates a higher degree of similarity.

Cosine similarity is widely employed in numerous cases due to its suitability for comparing vector directions. For instance, in literature [33], the K-Nearest Neighbor (KNN) algorithm was implemented with the Cosine similarity in a content-based recommender system, resulting in improved accuracy and reduced complexity. Literature [34] proposed an approach that integrates Cosine similarity into conventional classifiers, aiming to enhance text classification. And, literature [35] introduced a multi-view cosine similarity learning methodology for efficient face verification using multi-view information.

4) Comparison among the similarity metrics: According to the aforementioned statement, Euclidean Distance is widely employed as a similarity metric that quantifies the straight-line distance and primarily focuses on the magnitude of vectors. Dot Product represents the sum of element-wise products between corresponding positions in two vectors, considering both their direction and magnitude. Cosine similarity calculates the cosine of the angle between two vectors, placing emphasis on their directional alignment.

The analysis indicates that Euclidean Distance is suitable for continuous variables, while Cosine is particularly effective for text data or high-dimensional data. Besides, Dot Product excels in measuring similarity in vector spaces. These distinctions are summarized in Table II.

TABLE II
THE SENSIBILITY OF SIMILARITY METRICS AND THE LENGTH AND DIRECTION OF VECTORS

Similarity Metric	Length	Direction	Application
Euclidean Distance	√		Clustering, Classification
Dot Product	√	√	Recommended System
Cosine		√	Text Categorization

IV. Comparison among Vector Databases

In fact, selecting an appropriate vector database in business enterprises is greatly influenced by a multitude of factors, such as availability, reliability, and compatibility. This section presents three key criteria for evaluating vector databases and introduces three prominent vector databases.

A. Essential criteria when selecting

- **Availability:**
  The abundance of information in datasets necessitates a distributed architecture with sharding to ensure high availability in vector databases. Unlike traditional databases that use primary keys for sharding, distributed vector databases utilize vector similarity to enhance search and querying capabilities to facilitate the storage, retrieval, and analysis of extensive volumes of data.
- **Reliability:**

The reliability of vector databases is crucial in ensuring that users can CRUD (create, read, update, and delete) target vectors and audit logs without any data loss during database changes. This reliability can greatly facilitate precise querying of the vector database and the optimization of user experience.

* **Compatibility:**
  The compatibility of vector databases should be evaluated by cases and requirements. It is vital to verify its integration with the current programming environment and programming languages. Besides, pay attention to design schemes of Application Programming Interface (API) and Software Development Kit (SDK) that can impact the efficiency of database development.
* **Scalability:**
  The exceptional scalability enables vector databases to handle high-dimensional complex data and support high concurrent access. This is achieved through a distributed architecture and parallel computing technology, ensuring efficient storage and querying of the data.

B. Mainstream vector databases

Echoing the aforementioned rules, there are three mainstream solutions for the analysis of "Big Data" and they are Pinecone, Chroma, and Milvus, all equipped with distributed infrastructure. Table III provides a thorough overview of their fundamental characteristics across various scales.

TABLE III
BASIC PROPERTIES OF PINECONE, CHROMA, AND MILVUS

Name	Pinecone[6]	Chroma[7]	Milvus[8]
Euclidean Distance	A managed, cloud-native vector combining state-of-the-art vector search libraries and advanced features.	An AI-native open-source vector database designed to make it easy to build LLM applications.	An open-source vector database built to power embedding similarity search and AI applications.
Developer	Pinecone Systems Inc	Chroma Inc	Zilliz
Website	www.pinecone.io	milvus.io	www.trychroma.com
Initial Release	2019	2022	2019
License	Commercial	Apache-2.0	Apache-2.0
Cloud-based Only	Yes	No	No
Sharding	Yes	Yes	Yes
Index Types	Proprietary	HNSW	HNSW
Supported Programming Languages	Python	Python, Typescript	C++, Go, Java,Python

Pinecone is a closed-source vector database, while Chroma and Milvus are both open-source. Each of these three databases possesses its own distinct characteristics in terms of features. The following clarifies the primary distinctions in functionality between them:

* **Transparency:**
  Pinecone is designed for easy creation and deployment of large-scale machine learning applications in the cloud. It simplifies infrastructure complexities, allowing developers to focus on application development while enabling single-step filtering and real-time analytics.
* **Support for LLM:**
  Chroma supports both cloud and on-premises deployments, providing developers and organizations with the necessary resources to build applications based on LLM. Additionally, it excels in audio processing capabilities.
* **Integration:**
  The distinguishing feature of Milvus is its seamless integration with popular frameworks like PyTorch and TensorFlow, enabling easy deployment on both cloud and on-premises environments, making it an excellent choice for data science and machine-learning applications.

Reconsidering the selection criteria for vector databases and evaluate respective features of Pinecone, Chroma and Milvus, there are subtle nuance between them: **(1)** Pinecone, with robust distributed computing capabilities and scalability, has user-friendly APIs and is commonly used for real-time similarity search scenarios such as recommendation systems and personalized ads. However, it should be noted that this service comes at a relatively high cost. **(2)** Chroma enhances reliability and analytics capabilities, making it popular in domains like medical imaging, financial risk control, and other data-intensive scenarios. However, GPU acceleration is not supported. **(3)** Milvus, developed exclusively in Golang with exceptional scalability, provides a versatile client interface in multiple languages and GPU acceleration, widely used for vector retrieval scenarios like image and video retrieval. However, it has limited data analysis capabilities and lacks compatibility and collaboration with other database products.

V. POTENTIAL FUTURE DEVELOPMENTS IN VECTOR DATABASES

After exploring the fundamental technology and current status, it is imperative to delve deeper into future development trends and potential application domains of vector databases.

Firstly, the optimization of similar search algorithms and similarity metrics are essential to efficiently and accurately process various types of data, with the increase in data size and dimension. For instance, deep learning techniques can be utilized to generate vector representations.

Besides, the diversity of user requirements and application scenarios drives vector databases to provide more flexible functions. Especially, it is innovative to devise a user-friendly and succinct query language that caters to complex and dynamic query demands like reversing K-rank queries on vectors.

Moreover, the further popularization of LLM facilitates the integration with vector databases, thereby enhancing LLM's memory capabilities and expanding the application scope of vector databases.

In short, vector databases are specialized database systems designed for the storage and processing of vector data, offering support for high-dimensional data, efficient multi-modal queries, as well as robust distributed storage and computing capabilities. With the advancements in theoretical and practical

[6] https://www.pinecone.io/
[7] https://docs.trychroma.com/
[8] https://milvus.io/

exploration, vector databases will remain a subject of ongoing attention and refinement.

VI. Conclusion

In this paper, our primary focus is on surveying vector databases that have emerged in response to the unprecedented rates at which high-dimensional data are being generated. Vector databases are designed to outperform traditional ones and can rapidly retrieve approximate results through indexing and querying. By leveraging similarity search algorithms like K-Means, LSH, HNSW, and PQ along with metrics such as Euclidean Distance, Dot Product Similarity, and Cosine Similarity, vector databases can enhance search efficiency and reduce computational overhead. Physically, factors like availability, reliability, and compatibility should be considered when selecting a preferred vector database. Therefore, we introduced three mainstream options including Pinecone, Chroma, and Milvus, while also providing an outlook on the field's development.

References

[1] P. Maddigan and T. Susnjak, "Chat2vis: Generating data visualisations via natural language using chatgpt, codex and GPT-3 large language models," *IEEE Access*, 2023.

[2] K. Shen, Z. Ju, X. Tan, Y. Liu, Y. Leng, L. He, T. Qin, S. Zhao, and J. Bian, "Naturalspeech 2: Latent diffusion models are natural and zero-shot speech and singing synthesizers," *arXiv preprint arXiv:2304.09116*, 2023.

[3] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. of the IEEE international conference on computer vision*. IEEE Computer Society, pp. 2242–2251.

[4] D. O. Eke, "Chatgpt and the rise of generative ai: threat to academic integrity?" *Journal of Responsible Technology*, vol. 13, p. 100060, 2023.

[5] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.

[6] S. Murugesan and A. K. Cherukuri, "The rise of generative artificial intelligence and its impact on education: The promises and perils," *Computer*, vol. 56, no. 5, pp. 116–121, 2023.

[7] J. Martins, J. S. Cardoso, and F. Soares, "Offline computer-aided diagnosis for glaucoma detection using fundus images targeted at mobile devices," *Computer Methods and Programs in Biomedicine*, vol. 192, p. 105341, 2020.

[8] H. Chen, H. Shuai, and W. Cheng, "A survey of artificial intelligence in fashion," *IEEE Signal Processing Magazine*, vol. 40, no. 3, pp. 64–73, 2023.

[9] X. Zhou, C. Chai, G. Li, and J. Sun, "Database meets artificial intelligence: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 3, pp. 1096–1116, 2020.

[10] E. Basar, A. Saikia, and L. Saikia, "A survey on database in cloud computing with reference to the traditional database," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 7, p. 150, 2017.

[11] L. P. Osco, E. L. d. Lemos, W. N. Gonçalves, A. P. M. Ramos, and J. Marcato Junior, "The potential of visual chatgpt for remote sensing," *Remote Sensing*, vol. 15, no. 13, p. 3232, 2023.

[12] N. DeCastro-García, Á. L. M. Castañeda, M. F. Rodríguez, M. V. Carriegos *et al.*, "On detecting and removing superficial redundancy in vector databases," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–14, 2018.

[13] K. Z.-P. K. Fang, "A topology-concerned spatial vector data model for column-oriented databases," *International Journal of Database Theory and Application*, 2017.

[14] S. Singla, A. Eldawy, T. Diao, A. Mukhopadhyay, and E. Scudiero, "Experimental study of big raster and vector database systems," in *Proc. of 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, pp. 2243–2248.

[15] X. Zhang and Z. Ma, "Fuzzy rdf knowledge graph embeddings through vector space model," *IEEE Transactions on Fuzzy Systems*, vol. 31, no. 3, pp. 835–844, 2022.

[16] Y. Liu, F. Wu, C. Lyu, X. Liu, and Z. Liu, "Behavior2vector: Embedding users' personalized travel behavior to vector," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 8346–8355, 2021.

[17] K. A. Lee, Q. Wang, and T. Koshinaka, "Xi-vector embedding for speaker recognition," *IEEE Signal Processing Letters*, vol. 28, pp. 1385–1389, 2021.

[18] C. Lohrmann and P. Luukka, "A novel similarity classifier with multiple ideal vectors based on k-means clustering," *Decision Support Systems*, vol. 111, pp. 27–37, 2018.

[19] M. Ghayoumi, M. Gomez, K. E. Baumstein, N. Persaud, and A. J. Perlowin, "Local sensitive hashing (LSH) and convolutional neural networks (cnns) for object recognition," in *Proc. of 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, December 17-20, 2018*. IEEE, pp. 1197–1199.

[20] Y. Tian, X. Zhao, and X. Zhou, "Db-lsh 2.0: Locality-sensitive hashing with query-based dynamic bucketing," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[21] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Communications of the ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[22] Q. Yang, H. Ji, Z. Xu, Y. Li, P. Wang, J. Sun, X. Fan, H. Zhang, H. Lu, and Z. Zhang, "Ultra-fast and accurate electron ionization mass spectrum matching for compound identification with million-scale in-silico library," *Nature Communications*, vol. 14, no. 1, p. 3722, 2023.

[23] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[24] J.-H. Kim, Y.-R. Park, J. Do, S.-Y. Ji, and J.-Y. Kim, "Accelerating large-scale graph-based nearest neighbor search on a computational storage platform," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 278–290, 2022.

[25] J. Heo, Z. L. Lin, and S. Yoon, "Distance encoded product quantization for approximate k-nearest neighbor search in high-dimensional space," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 9, pp. 2084–2097, 2019.

[26] J. Fan, Z. Pan, L. Wang, and Y. Wang, "Codebook-softened product quantization for high accuracy approximate nearest neighbor search," *Neurocomputing*, vol. 507, pp. 107–116, 2022.

[27] M. Coscia, A. Gomez-Lievano, J. Mcnerney, and F. Neffke, "The node vector distance problem in complex networks," *ACM Computing Surveys*, vol. 53, no. 6, pp. 1–27, 2020.

[28] L. H. Lee, C. H. Wan, R. Rajkumar, and D. Isa, "An enhanced support vector machine classification framework by using euclidean distance function for text document categorization," *Applied Intelligence*, vol. 37, no. 1, pp. 80–99, 2012.

[29] M. Djellab, N. Mehallegue, and A. Achi, "Use of neumann series decomposition to fit the weighted euclidean distance and inner product scoring models in automatic speaker recognition," *Pattern Recognition Letters*, vol. 125, pp. 500–507, 2019.

[30] L. Naveen, A. Raj, S. Rajalakshmi *et al.*, "Abstractive text summarizer: A comparative study on dot product attention and cosine similarity," in *Proc. of 2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, pp. 1–8.

[31] Z. Hu and A. G. Bors, "Dot-product based global and local feature fusion for image search," in *Proc. of 2022 IEEE International Conference on Image Processing, ICIP 2022, Bordeaux, France, 16-19 October 2022*. IEEE, pp. 1911–1915.

[32] H. Zhou, Y. Li, and X. Miao, "Low-time-complexity document clustering using memristive dot product engine," *Science China Information Sciences*, vol. 65, no. 2, p. 122410, 2022.

[33] R. H. Singh, S. Maurya, T. Tripathi, T. Narula, and G. Srivastav, "Movie recommendation system using cosine similarity and knn," *International Journal of Engineering and Advanced Technology*, vol. 9, no. 5, pp. 556–559, 2020.

[34] K. Park, J. S. Hong, and W. Kim, "A methodology combining cosine similarity with classifier for text classification," *Applied Artificial Intelligence*, vol. 34, no. 5, pp. 396–411, 2020.

[35] Z. Wang, J. Chen, and J. Hu, "Multi-view cosine similarity learning with application to face verification," *Mathematics*, vol. 10, no. 11, p. 1800, 2022.