Approximate Nearest Neighbor Search in High Dimensional Vector Databases: Current Research and Future Directions

Yao Tian[†], Ziyang Yue[‡], Ruiyuan Zhang[†], Xi Zhao[†], Bolong Zheng[‡*], Xiaofang Zhou[†*]
[†]The Hong Kong University of Science and Technology, Hong Kong SAR, China
[‡]Huazhong University of Science and Technology, Wuhan, China
{ytianbc,zry,xzhaoca,zxf}@cse.ust.hk, {ziyangyue,bolongzheng}@hust.edu.cn

Abstract

Approximate nearest neighbor search is an important research topic with a wide range of applications. In this study, we first introduce the problem and review major research results in the past. We then discuss the current work in the database research community, categorizing the work by their key underlying methodologies, such as locality-sensitive hashing, product quantization, and approximate nearest neighbor graphs. Finally, we examine several new directions, with a focus on vector databases to support large language models.

1 Introduction

1.1 Dense Vector and Searching

We are witnessing a notable transition towards vectorized data representations, emerging as the go-to method for encapsulating diverse data forms, from text and images to videos. This transformation is deeply rooted in the successes of deep neural network-based representation learning. The value of such vector representations is further accentuated by the recent breakthroughs of large language models like ChatGPT and the rapid strides in the multi-modal domain. Specifically, dense vectors representations have gained traction across diverse sectors, including recommendation systems, search engines, and e-commerce, etc.

To understand the significance of dense vector representations, it is essential to differentiate them from sparse vectors. While sparse vectors contain bits of information distributed sparsely, dense vectors contain the compressed information across every dimension, making them more information rich. Such distinction can be likened to the difference between syntax and semantics in natural language processing. Sparse vectors allow for syntax-based comparisons of sequences, which is efficient in storage. Even if two sentences differ in meaning but share the same syntax, sparse vectors might closely match them. In contrast, dense vectors can be seen as numerical encodings of semantic meaning, which contain their abstract meaning and relationships. Consequently, searches on dense vectors that are derived from text, can be treated as semantic-based searches. This stands in contrast to the traditional syntax-based searches that are typically conducted on sparse vectors.

The adoption of dense vector representations offers clear benefits, including simplicity of data representation and increased computational efficiency. The utility is particularly evident in foundational computational tasks such as search, classification, and clustering, which underpin more complex challenges. Among all operations on dense vectors, the nearest neighbor search, which aims to locate the closest point to a specific reference point, stays in the central position. Unfortunately, it is often infeasible to retrieve the exact nearest neighbors of the query point due to a phenomenon known as "the curse of dimensionality". As a result, approximate nearest

*The authors are listed in alphabetical order. Bolong Zheng and Xiaofang Zhou are co-corresponding authors.

neighbors (ANN) search algorithms have been designed to retrieve the neighbors that are close enough to the reference point. The goal of an ANN search algorithm is to retrieve approximate nearest neighbors of the query point in a low response time, which results in the accuracy-vs-efficiency trade-off [54].

ANN search is a well-established problem that has garnered significant research attention. The computational complexity is determined by the number of data points (n) and the dimensionality (d), resulting in a complexity of $O(nd)$. Many existing methods aim to address this challenge by focusing on either reducing the number of data points to examine or decreasing the dimensionality. In terms of reducing the number of data points to examine, two widely employed approaches are space partitioning and proximity graph. Notable methods falling within this category include the kd-tree, Approximate Nearest Neighbors Oh Yeah (ANNOY) [68], and the Hierarchical Navigable Small World (HNSW) algorithm [69]. On the other hand, in the realm of dimensionality reduction, quantization and hashing techniques have gained considerable popularity. Representative methods in this category include Locality-Sensitive Hashing (LSH) [3], Spectral Random Sampling (SRS) [70], and Product Quantization (PQ) [35].

1.2 Vector Search Libraries and Vector Database

The growing need for high-dimensional ANN search in areas such as social media, e-commerce, and digital advertising has shifted the focus from individual algorithms to more integrated libraries. A leading example is Faiss [42], also known as Facebook AI Similarity Search. Faiss is a C++ library with Python bindings that tailored for the efficient clustering and search of dense vectors. The library encompasses a broad range of similarity metrics and is equipped with widely recognized methods for ANN search, including but not limited to IVFADC [35], HNSW [58], and LSH [3]. Notably, Faiss has several GPU-optimized algorithms and seamlessly supports the IVF-PQ series on both CPU and GPU platforms [56]. Alongside Faiss, libraries such as ANNOY and Non-Metric Space Library (NMSLIB) provide similar toolkits. ANNOY [68] is utilized by Spotify, while NMSLIB [43] has been integrated into Amazon's Elasticsearch Service. Additionally, Alibaba Cloud has introduced a vector analysis framework within their AnalyticDB for PostgreSQL, designed to fetch unstructured data and facilitate association analysis between unstructured and structured data sets.

While libraries like Faiss offer significant capabilities, they might not fully address the complexities of real-world applications. Recognizing this limitation, the concept of vector databases emerges. They are designed much like traditional relational databases but specifically for vector management. They not only offer efficient data indexing, storage, and filtering of vector attributes but also bolster distribution, parallel processing, and facilitate real-time data and index updates. Parallel to the features of a conventional DBMS, they prioritize data safety with backup and collection functionalities. The adaptability is evident in their effortless integration with various data processing tools, analytics platforms, visualization instruments, and AI plugins, enhancing the overall data management workflow. Moreover, these databases are equipped with robust security features and access controls, ensuring the protection of sensitive data, an aspect sometimes overlooked in standalone vector indices. To sum it up, vector databases refine and strengthen the data management landscape with their advanced security and integration capabilities.

In the vector database domain, Pinecone [75], Milvus [76], and Weaviate [77] emerge as frontrunners. While all three offer robust solutions for storing, indexing, and searching vast datasets, Pinecone differentiates itself as a closed-source platform. It is exclusively available as a SaaS service, with all user interactions channeled through its API. Conversely, both Milvus and Weaviate are open-source vector databases, benefiting from the collaborative efforts of a varied mix of companies and individual contributors. Some of these participants also extend specialized commercial services and support.

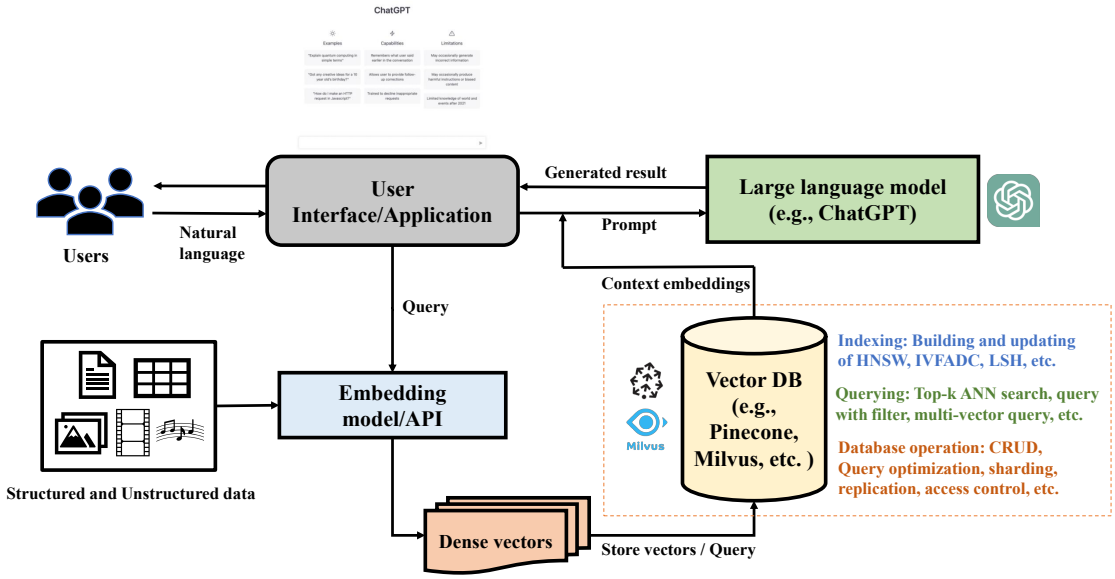


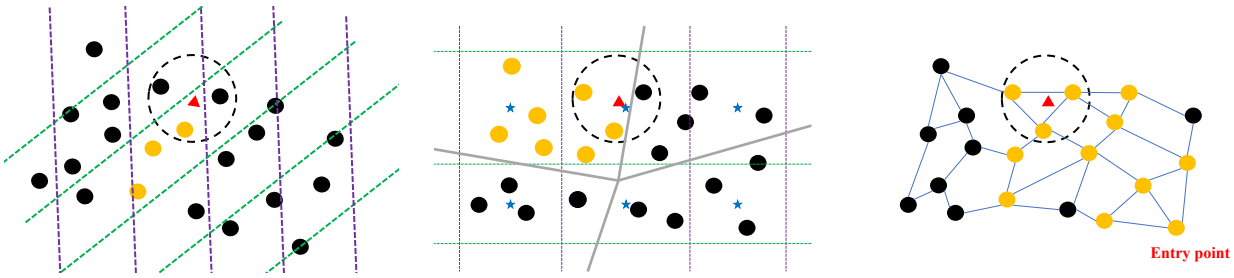
Figure 1: Framework for Enhancing Answer Quality in LLM-based Chatbots using Vector Databases

1.3 Vector Database for LLM

With the development of large language models such as ChatGPT, chatbots have become more advanced and can now be used for a wide range of applications [72–74]. While they are effective at accurately retrieving information, they face challenges that can be resolved through vector databases. First, searches on dense vectors derived from text are mainly based on semantics. Vector databases use index structures for ANN search, which enables them to efficiently locate the most semantically relevant information. This allows chatbot models to access a knowledge base where context can be stored for extended periods in a memory-efficient manner. Second, incorporating new data into LLMs without costly retraining is a challenge since current LLMs are static. However, many vector indexes are designed to manage dynamic datasets [60], which can help extract key information from these datasets when training a new LLM from a trained old one. Finally, data exists in multiple types, and LLMs as a language model do not learn enough about other types of data besides text. Vector databases can help manage these multimodal data in the unified embedding space and feed them to the LLMs. This approach would be a crucial step for LLMs to process various kinds of data sources. Figure 1 shows a framework for enhancing answer quality in LLM-based chatbots. This framework utilizes a vector database as a multi-functional component, serving as a cache, extended memory, and external knowledge base. The vector database not only offers essential indexing and ANN search services but also incorporates comprehensive functionalities expected from a database, including sharding, access control, and query optimization. By leveraging the vector database, the chatbot gains the ability to efficiently retrieve and store relevant information, leading to improved answer quality and enhanced performance.

1.4 Contribution

In this study, our objectives are manifold. 1) We delve into the historical trajectory and evolution of the ANN search problem. Based on our research results, we provide a comprehensive overview of the state-of-the-art methods that have been developed. We discuss the foundational principles, key milestones, and the most recent advancements in ANN search. 2) We transition into the challenges that ANN search methods may encounter in the



(a) LSH-Based Method (b) Quantization-Based Method (c) Graph-Based Method

Figure 2: Difference in the LSH-based method, graph-based method and PQ-based method. The red triangle is the query point q. The three points in the black dashed circle are the 3NN of q. The orange points denote those that are accessed during the search.

burgeoning era of large language models (LLMs). As the application of ANN search becomes more intertwined with LLMs, a myriad of complexities arise. These challenges span from issues of scalability, where the sheer volume of data can overwhelm traditional methods, to the intricacies of parallel computation, which demands efficient synchronization and communication mechanisms. Furthermore, the need for on-disk indexing introduces another layer of complexity, necessitating efficient data retrieval methods that minimize latency. Additionally, as with any data-centric application, data security remains paramount, prompting rigorous measures to ensure data integrity and confidentiality. 3) We conclude by projecting forward, offering our insights into the potential future research directions in the ANN search landscape. This will encompass emerging trends, potential breakthroughs, and areas that warrant deeper exploration in light of the challenges and opportunities presented by LLMs.

2 Notation and Preliminaries

We start by clearly defining ANN search. Then, we introduce key index structures used in high-dimensional ANN search. We cover seminal methods such as Locality-Sensitive Hashing (LSH), Product Quantization (PQ), and Hierarchical Navigable Small World (HNSW). The aim is to arm the readers with essential terminologies and foundational concepts, paving the way for the comprehensive discussions that ensue.

2.1 Problem Definition

We take (c, k)-ANN search in the Euclidean space as an example to illustrate the ANN search problem in high-dimensional spaces. Let $\mathcal{D}$ be a set of points in d-dimensional Euclidean space $\mathbb{R}^d$ with cardinality $|\mathcal{D}| = n$ and $\|o_1, o_2\|$ denote the Euclidean distance between points $o_1, o_2 \in \mathcal{D}$.

Definition 2.1 ((c, k)-ANN Search [19]) *Given a query point q, an approximation ratio $c > 1$ and a positive integer k, a (c, k)-approximate nearest neighbor search returns k points $o_1, \ldots, o_k$ that are sorted in ascending order w.r.t. their distances to q. If o_i^* is the i-th nearest neighbor of q in $\mathcal{D}$, it satisfies that $\|q, o_i\| \leq c \cdot \|q, o_i^*\|$.*

We denote the (c, k)-ANN search with $k = 1$ as c-ANN search. An LSH-based method can ensure a correct c-ANN with high probability [5, 19]. While in the graph-based and quantization-based methods, we usually do not explicitly use c but use *recall* to control the query quality where *recall* measures how many exact kNN results is found during the search.

2.2 Seminal Algorithms for High-dimensional ANN Search

2.2.1 Locality-Sensitive Hashing

Among solutions to the high-dimensional ANN search problem, locality-sensitive hashing (LSH) is known for its sub-linear query time and robust theoretical guarantee of query accuracy. The basic idea behind LSH is to map data points into buckets using a set of hash functions such that nearby points in the original space have a higher probability of being hashed into the same bucket than those far-apart points. As shown in Figure 2(a), two groups of parallel lines (purple and green lines) act as two hash functions dividing the space into several parallelogram-shaped buckets. When conducting the ANN search, we check the points in the bucket where the query point falls. A hash function with such locality-preserving property is called locality-sensitive hashing. LSH was first proposed in [1, 2] for the Hamming distance and later extended to several more popular distances, such as the Euclidean distance. E2LSH [3] is a seminal algorithm for ANN search in the Euclidean space. It adopts the p-stable distribution-based function proposed in [4] as its LSH function. The algorithm concatenates a set of $K (K \ll d)$ independent LSH functions to form a hash table, which is then repeated L times to generate L K-dimensional hash tables. Two points in the original space are considered a collision if they are mapped into the same bucket at least once. Intuitively, the probability of two different points being hashed into the same bucket increases (or decreases) with K (or L). Theoretical results validate that, with properly chosen values for K and L, E2LSH can solve the ANN search problems in sub-linear time with a constant success probability. However, E2LSH needs to prepare a large number of hash tables, which causes large storage costs. Additionally, points close to the query points may be partitioned into buckets different from the one containing the query point, especially when the query is near the bucket boundaries, which jeopardizes the accuracy.

2.2.2 Product Quantization

Product Quantization (PQ) [35] is a widely-used vector quantization technique. It aims to compress high-dimensional vectors into short codes to reduce the space overhead in ANN search. It first divides the d dimensions equally into M groups. Thus each vectors are divided into M sub-vectors and the whole space is divided into M orthogonal d/M-dimensional sub-spaces. Then, in each sub-space, sub-vectors are clustered into K groups, and each sub-space centroid is encoded by a single integer code from 1 to K. Finally, each vector can be identified by the concatenation of codes of the nearest sub-space centroid in each sub-space, and the concatenation of the these sub-space centroids is the corresponding quantized vector. In search procedure, the Euclidean distance is replaced by the asymmetric distance as an approximation, which is the Euclidean distance between the query and the quantized vector. The asymmetric distance can be efficiently computed by table lookup and addition of sub-space distances, after the distances between sub-space centroids and sub-vector of query in each sub-space are computed and saved in a distance table. IVFPQ is a variant of PQ, which employs inverted file (IVF) index to index vectors before quantization. Vectors are clustered into different cells, and each IVF list corresponds to one cell and stores the identities of all vectors in it. During ANN search, only vectors located in a few nearest cells are evaluated. Therefore, IVFPQ achieves higher efficiency. As shown in Figure 2(b), the 2D space is divided into two orthogonal 1D sub-spaces with 2 and 3 centroids respectively, thus forming 6 quantizers (the blue stars) in the whole space. Vectors are clustered into 4 cells, and only the nearest one is evaluated during ANN search.

2.2.3 Proximity Graph

Graph-based methods have recently come to the forefront as a superior approach for ANN search. They leverage the power of proximity graphs (PG) and have demonstrated superior performance in terms of both accuracy and efficiency [58, 59, 62]. The fundamental structure of graph-based methods is a proximity graph, represented as $G = (V, E)$. Here, the vertex set V symbolizes all data points in the dataset $\mathcal{D}$, while the edge set E encompasses all edges between vertices if the corresponding points are sufficiently proximate in the original space. As it is

computationally challenging to identify neighbors for each vertex, the construction cost of an exact proximity graph escalates to a minimum of $O(n^2)$ distance computations. This is prohibitively expensive for large-scale datasets. So, even though many well-designed graph structure has been used for nearest neighbor search in multi-dimensional space several decades ago, such as MRNG [53], Delaunay Graph and NN graph, they are hardly adaptive for high-dimensional ANN search problem due to the high indexing cost. Hence, existing graph-based methods concentrate on devising more efficient strategies to construct an approximate proximity graph (APG), such as approximate Delaunay Graph and approximate NN graph, while ensuring robust query performance. As shown in Figure 2(c), each data point in graph has 2-4 neighbors. The query begins from the bottom right point and approaches to the correct results via greedy search in the APG.

3 Current Research Activities

We proceed to introduce the evolutionary trajectories of the above-mentioned representative categories of indexing structures for ANN search. For each category, we trace its development from inception to its current state, highlighting key milestones and innovations. Additionally, we spotlight the state-of-the-art studies in each category, offering a comprehensive view of the present landscape.

3.1 Locality-Sensitive Hashing based Methods

To remedy the issues with space and hash boundaries in seminal work, a range of methodologies have been ramified based on different indexing frameworks, alternative search strategies, and elaborate hash functions. One such method is Multi-Probe [6], which proposes examining close-by buckets along with the one containing the query point, following an ascending distance score. Rather than utilizing hash tables, LSB-forest [5] leverages the Z-order curve and B+-trees to index the projected spaces and finds candidates by finding the data point with the greatest Length of the Longest Common Prefix (LLCP). LCCS-LSH [7] introduces the concept of the Longest Circular Co-Substring (LCCS) and a data structure Circular Shift Array (CSA), with which the candidates are identified by the largest LCCS. These methods carry not only theoretical guarantees of accuracy and efficiency in E2LSH, but also reduce the space cost. The techniques of Collision Counting and Virtual Rehashing, introduced in C2LSH [8], offer further space optimization at the cost of query efficiency. Based on the intuition that the incidence of collisions among nearby points tends to exceed that of points distant from each other, C2LSH relaxes the collision criteria from exactly K collisions to any l collisions, where $l < K$ is a given value. Consequently, C2LSH manages to maintain solely K one-dimensional hash tables, as opposed to the original L K-dimensional hash tables. Building upon C2LSH, QALSH [9] introduces a query-aware LSH function. By constructing dynamically evolving query-centric buckets, QALSH mitigates the hash boundary issue. R2LSH [10] enhances the performance of QALSH by projecting data onto multiple two-dimensional spaces rather than one-dimensional projections. VHP [11] treats QALSH's buckets as hyperplanes, and introduces the concept of a virtual hypersphere to achieve a reduced space complexity. I-LSH [12] and EI-LSH [13] introduce an incremental search strategy and a set of adaptive early termination conditions. This strategy enables incremental access to data pointss based on their projected distances and early stops the query process if a good enough result is found. Upon LSH's locality-preserving property, SRS [14] and PM-LSH [15] propose leveraging distances between two points in projected spaces to estimate their corresponding distances in the original space. This enables the determination of ANNs in the original space via lots of exact nearest neighbor searches in projected spaces. These methods rely on a sole multi-dimensional index, yielding gains in space efficiency. However, it's worth noting that the query costs of C2LSH, SRS, and their variants no longer maintain a sub-linear advantage. Concerning hash functions, random linear projections stand out as the prevailing LSH function, while a multitude of studies continue their dedicated efforts to innovate and enhance hash functions and strategies [16–18, 21].

3.1.1 DB-LSH: Locality-Sensitive Hashing with Query-based Dynamic Bucketing

DB-LSH [19] is a state-of-the-art work that can achieve the lowest query time complexity to date. It organizes the projected spaces with multi-dimensional indexes instead of fixed-width hash buckets, which significantly reduces space costs. During the query phase, DB-LSH dynamically constructs query-centric buckets with the required widths and conducts multi-dimensional window queries to efficiently generate candidates. Different from other query-centric methods, the buckets in DB-LSH are still multi-dimensional cubes like in E2LSH, making it possible to not only generate high-quality candidates but also to achieve sub-linear query cost. Furthermore, DB-LSH achieves a much smaller bound on query cost compared to existing work, while using an proper and practical bucket size. Rigorous theoretical analysis and extensive experiments show that DB-LSH outperforms the existing LSH methods significantly for both efficiency and accuracy. DB-LSH 2.0 [20] is an extension work of DB-LSH, including DBI-LSH and DBA-LSH. Instead of exponentially enlarging the radius of query window, DBI-LSH always probes the next best point in L projected spaces. Such a strategy makes the search terminate at a more proper search radius, and thus can achieve better accuracy and efficiency. Recognizing the wide variance in the number of candidates required to reach a given accuracy across queries, DBA-LSH is developed on DBI-LSH. This variant incorporates several adaptive early termination conditions by leveraging the intermediate query information, which aggressively reduces the number of points accessed. It has been proven that DBA-LSH can achieve a faster search without breaking the theoretical guarantee.

In summary, LSH-based methodologies offer fast query processing and probabilistic theoretical guarantees of query accuracy. Moreover, due to their straightforward architecture, LSH-based indexes exhibit quicker construction, streamlined support for updates, and notably smaller index sizes in comparison to alternative approaches for ANN search. Recently, studies adopt the LSH framework to solve other kinds of queries, such as maximum inner product search (MIPS) [22] and point-to-hyperplane NN search [23] in high-dimensional spaces. These examples demonstrate the superior performance and great flexibility of LSH. Nevertheless, as evidenced by various studies [24], despite their theoretical assurances, LSH-based methods frequently encounter difficulties in outperforming proximity graph-based and product quantization-based techniques in terms of practical accuracy, a topic that will be delved into in the upcoming sections.

3.2 Quantization based Methods

Although PQ reduces the space overhead, the process of compression is lossy. The error incurred when vectors are approximated by their quantized vectors, which is called quantization distortion. This results in inadequate search accuracy. Furthermore, due to the coarse-grained structure of IVF index, its effect of pruning deteriorates on large datasets. Therefore, studies on PQ mainly falls in two directions, thus reducing quantization distortion for higher recall [33, 34, 36, 38] and pruning candidates to evaluate for higher efficiency [29, 30, 37, 41].

OPQ [33] and LOPQ [36] attempt to reduce the quantization distortion by introducing rotation matrix. OPQ adopts a global orthogonal matrix to optimize space decomposition. The original space is first transformed by the matrix, and then decomposed in the way PQ does. On the contrary, LOPQ adopts multiple local orthogonal matrices, in order to better fit data with strongly multi-modal distribution. It indexes and partitions vectors with coarse quantizers, and applies a local rotation transformation on residual vectors in each partition. DPG [34] uses additional bits to quantize the distance between the vector and corresponding quantizer, because quantization distortion increases when this distance is large. Distance quantizers serve as an error correction term in distance estimation, and improve the search accuracy. Noh et al. [38] propose a jointly optimization mechanism of coarse and fine quantizers. The coarse quantizers are initialized by K-means, and then optimized with fine quantizers iteratively. In each step, fine quantizers are trained, and each coarse quantizer is updated by the mean error of corresponding vectors.

Some studies are devoted to design more fine-grained index structure to reduce the number of candidates. IMI [29] employs the idea of PQ and leverages orthogonal space decomposition to generate inverted indices. Therefore

the one-dimensional header in inverted list becomes a multi-dimensional one, and finer partitions are achieved. GNO-IMI [30] adopts a non-orthogonal space decomposition method, because orthogonal decomposition cannot fit data with significant correlations between dimensions. It uses two-order clustering to cluster the original vectors and residual vectors respectively, so that each centroid can be represented by the weighted sum of corresponding centroids from two order. Li et al. [37] propose a learned adaptive early termination mechanism in IVFPQ. The gradient boosting decision tree model is build and trained, which takes the query vector and intermediate search results as input features, and decides online whether to early terminate search.

3.2.1 Probing Cardinality Estimation IVFPQ

In IVFPQ and its variants, an input parameter *nprobe* is required. It determines the nearest number of *nprobe* cells to probe, and thus controls the trade-off between accuracy and efficiency. However, it turns out that there is no such a single *nprobe* that is optimal for all queries. Intuitively, if a query locates at the center of a cell, its nearest neighbors may all fall in this cell, so that an $nprobe = 1$ is enough. On the contrary, if a query locates on the boundary of several cells, we need to set *nprobe* to a large number to include its nearest neighbors distributed in different cells. A fixed *nprobe* will result in that some queries probe either redundant or insufficient cells. Therefore, *nprobe* is an "impossible-to-set" parameter that should be eliminated.

PCE-IVFPQ (Probing Cardinality Estimation IVFPQ) [41] formally defines and addresses the probing cardinality estimation problem for the first time. The probing cardinality estimation problem aims to learn a function to estimate a query-dependent minimum probing cardinality (i.e. *nprobe*) for a target recall, given a query vector, a target number of nearest neighbors, and distances between the query and all cell centroids. This problem is a regression problem and can be solved by deep learning techniques. However, using deep learning to estimate probing cardinality faces two main challenges. (1) Due to the sparsity of high-dimensional space, it is difficult to capture the distance distribution between the query vector and database vectors if applying a DNN directly. (2) The data distribution is imbalanced across cells. This results in that the number of vectors to evaluate varies significantly for different queries even with the same *nprobe*, which leads to poor estimation. For the first challenge, specialized modules are designed to process different features. PCE-Net consists of three encoder networks and one decoder network. The query vector, target number of nearest neighbors and distances to all centroids are encoded by three encoders respectively, and then concatenated to feed into the decoder. The decoder outputs the estimated probing cardinality. To mitigate the data distribution imbalance across cells, a hierarchical balanced clustering algorithm is designed, which can generate balanced cells efficiently. Besides, two additional optimization strategies are proposed and further reduce distance computations during cell probing.

In summary, PCE-IVFPQ reduces redundant candidate vectors greatly and achieves the state-of-the-art performance among IVFPQ-based methods w.r.t. search efficiency. The study of probing cardinality estimation is orthogonal to existing studies such as OPQ or IMI, and can be easily adopted by them.

3.3 Graph based Methods

Efficiently constructing an Approximate Proximity Graph (APG) is a significant challenge when designing a graph-based index. Various strategies have been proposed to address this challenge. NN-Descent is a prevalent technique for constructing approximate Nearest Neighbor (NN) graphs. This method involves building an Approximate Proximity Graph (APG) from a random graph, with the edges for each point updated iteratively through a local search among the query's close neighbors. The construction complexity of NN-Descent is reduced to $\tilde{O}(n^{1.14})$, making it significantly more efficient than brute-force methods. The NN-Descent algorithm is employed in numerous graph-based approaches, such as EFANNA [47], NSG [48] and others [57, 65]. Several derivatives of this algorithm have also been developed [44]. On the other hand, the Hierarchical Navigable Small World (HNSW) algorithm constructs its graph by sequentially inserting points. When inserting a point, denoted as o, the number of layers o should be placed in is determined through a random number. A query is then conducted

for o in the current index, and o's neighbors are chosen from the obtained results. Additionally, HNSW employs an edge occlusion rule to decrease the out-degree, a procedure proven to be identical to the one used in NSG. This rule enhances the distribution diversity of neighbors and boosts query efficiency. Known for its superior performance in addressing the ANN search problem, HNSW is implemented in several widely-used libraries like NMSLib [43] and Faiss [42], which offer efficient tools for similarity search.

While APG has emerged as the most promising method for solving ANN search problems, updating APGs for dynamic datasets can be quite challenging. As the dataset evolves, many points in the proximity graph need to be inserted and deleted. Inserting a point in the APG is relatively easy using the same strategy as the consecutive insertion strategy. However, deleting points in the APG is difficult as the point is connected to other points and many edges need to be dropped. There are two typical policies to address deletions in an APG [60]. The first policy is to drop graph vertices corresponding to deleted points, including their in-edges and out-edges. However, this policy can make some vertices connected to it become sparse, resulting in a degradation of graph quality. The second policy is that when deleting vertex o, for any pair of directed edges $(o_{in} \rightarrow o)$ and $(o \rightarrow o_{out})$ in the graph, we add the edge $(o_{in} \rightarrow o_{out})$ in the updated edge set of o_{in}. This policy addresses the problem in the former policy and ensures the graph quality does not degrade during updating, but heavily increases the deletion cost because it is time-consuming to update the edge sets of all o_{in}s. Therefore, ensuring graph quality and update efficiency when updating APGs is a challenging task.

3.3.1 Locality Sensitive Hashing-Approximate Proximity Graph

Locality Sensitive Hashing-Approximate Proximity Graph (LSH-APG) [67] is an innovative graph-based method that leverages lightweight LSH indexes to construct the APG and expedite ANN search processing. LSH indexes, while swiftly constructed, often fall short in terms of query accuracy. On the other hand, graph-based methods excel in query processing performance but are hindered by the high construction cost due to their intricate construction and edge selection strategies. LSH-APG aims to mitigate these limitations inherent in both LSH and graph-based methods. It utilizes LSH indexes to quickly retrieve preliminary query results as the starting point for a search in an APG, then employs graph-based techniques to further refine the accuracy of the query result. In a departure from HNSW and NSG, which reduce the number of edges based on the edge occlusion rule, LSH-APG introduces an accurate and scalable pruning strategy to filter out neighbors distant from the query point. This approach markedly reduces the number of points accessed during graph search, effectively boosting query efficiency without increasing the construction cost. To construct the graph index, LSH-APG employs the consecutive insertion strategy where LSH framework can help find the candidate neighbors. All points are consecutively incorporated into the APG, where each point is treated as a query point and inserted into the graph index based on its nearest neighbors. It addresses the issue of high construction cost with the assistance of the LSH framework. This strategy not only curtails construction cost by enhancing search efficiency via the LSH framework, but also allows for a formal correctness and complexity analysis of LSH-APG. The theoretical result show that LSH-APG have a nearly $O(n)$ query cost.

Furthermore, LSH-APG has designed a “mark-and-delete" policy to alleviate index update issues, which aims to strike a balance between graph quality and updating efficiency. To delete a point o in the graph indexes, LSH-APG first marks o and all its out-edges. Typically, only out-edges are stored in APGs, and finding in-edges is a challenging task that is not considered in the previous two policies. To address this, LSH-APG uses an approximate query algorithm to find the in-edges, with the query cost bounded by a given threshold C_{Dm} to control deletion efficiency. If an in-edge of o is not found in the search, it is left for deletion in subsequent searches. If it is found during an ANN search later, it is discarded, and the in-degree of o is decreased by one. Once the in-degree of o becomes zero, all its out-edges and o itself are discarded. To prevent in-edges from occupying space for an extended period, LSH-APG also traverses the graph and discards all edges to be deleted when their number reaches 10% of the total number of edges in LSH-APG. To ensure graph quality, LSH-APG controls the out-degree of each vertex to be within the interval $[T, 2T]$. When an edge (u, o) is marked for

deletion, LSH-APG checks whether u's out-degree decreases to less than T. If so, the number of u's neighbors is increased to $2T$ by finding points in the neighbors of u's neighbors. This approach reduces the negative impact of deletion on graph quality with an acceptable cost.

4 Challenges and Future Directions

4.1 Hybrid Queries

Efficient ANN search algorithms have profoundly shaped a multitude of applications. However, as data becomes increasingly complex and multifaceted, there arose a need to refine these searches further, leading to the evolution of hybrid queries. For instance, consider an e-commerce platform where users search for products using both textual queries and specific attributes like price range, brand, or manufacturing date. While the primary search is still for products similar to a given product using textual similarity as the distance metric, hybrid quires require refining results further by considering the filters applied, ensuring that the results not only match the textual query but also fall within the specified price range, belong to the chosen brand, or were manufactured within the given date range. This introduces an additional layer of complexity into traditional ANN search.

To address such hybrid queries, two straightforward ways are post-processing and pre-processing. The post-processing approach involves building a standard ANN search index, querying as usual, and then post-processing the results to select only those that align with the query filter. While simple, its performance is often found lacking in practical scenarios. Conversely, the pre-processing method involves building a distinct index for each possible filterable label, which can become infeasible with a large number of attribute constraints. Another intriguing method is inline-processing which integrates filter metadata with each vector into the index, and thus it simultaneously applies the filtering criteria as the search progresses through the dataset. Inline-processing can potentially offer more efficient results retrieval, but the state-of-the-art algorithms [26–28] have yet to reach satisfactory levels.

Optimizing algorithms for faster search, ensuring scalability for massive datasets, developing dynamic filtering mechanisms that can adapt to changing data landscapes, providing instant results for real-time applications like web search, and exploring the integration of deep learning techniques for enhanced accuracy are areas that need further improvement. Additionally, while the foundational concept remains intact, hybrid queries manifest in various forms, each customized to address specific application requirements. Some variants involve ANN searches with intricate predicates, while others demand the intersection of multiple ANN search results from distinct vector attributes. Some variants prioritize speed, while others might emphasize accuracy or the ability to handle dynamic datasets where data points can be added or removed frequently. All of these aspects remain open questions, beckoning further exploration and refinement.

4.2 Out-of-Distribution Queries

Out-of-distribution (OOD) queries refer to queries that are not drawn from the same distribution as the indexed data. A practical example of OOD is when a user searches through an image index using only a textual description as input. Even if both the image and text embeddings share the same representation space, the embeddings generated might lie in different distributions, leading to challenges in retrieval. State-of-the-art ANN search algorithms, such as graph-based and clustering-based indexes, achieve better query accuracy and efficiency over prior data-agnostic methods like LSH by employing data-dependent index construction. However, when the query data is OOD, there would be a significant performance decline due to the overfitting to the index data distribution. By utilizing a small sample set drawn from the query distribution apriori, [25] improves the mean query latency for OOD queries, but there is still much to explore and develop in this area. Handling OOD queries effectively is crucial for ensuring the robustness and reliability of AI systems, especially in critical applications. An ideal system should be able to generalize to OOD examples and flag those that are beyond its capability. It is worth

noting that OOD detection and handling are active areas of research, not only within the scope of ANN search, but also across various domains, spanning from computer vision to natural language processing.

4.3 Data Series Similarity Search

A data series is an ordered sequence of real numbers with length l. The most common type of data series is time series, where values are ordered by timestamp. This kind of data is exploding as the world is increasingly measured by sensors and other devices. Similarity search is a way to handle data series analysis task, such as anomaly detection, clustering and frequent pattern matching.

Data series can be treated as a special kind of high-dimensional vectors with ordered dimensions and high correlation between neighboring values. Similarly, exact and approximate k-nearest neighbors search problem can be defined for data series. However, except similarity metrics commonly used for high-dimensional vectors (Sec. 2), there exist a few distance functions specialized for data series. For example, Dynamic Time Warping (DTW) [40] automatically finds the correspondence between dimensions of two data series, which maximizes their similarity. Shape-based distance (SBD) [39] slides one data series over the other and uses the maximum cross-correlation to determine their similarity. These methods may outperform Euclidean distance at some scenarios since they offer better alignment, yet also incur more computation overhead. Nevertheless, Euclidean distance still remains one of the most popular similarity metrics.

Over the years, the data series community has proposed many methods for data series similarity search, which are seldom considered and studied together with those for high-dimensional vectors until recently [78–80]. Experiments conducted in [31, 32] compare the approaches from both communities and indicate that methods specialized for data series can achieve better performance. In addition, extensions are proposed to enable the existing data series indexes to support δ-ϵ-approximate search. It remains an open question if there exists a unified framework that can bridge the two worlds.

4.4 Scalability

As we continue to generate and collect massive amounts of data in various domains, the scalability of vector databases becomes a critical concern [56, 61, 80]. Consider the case of a global e-commerce platform, such as Taobao. Every day, Taobao collects vast amounts of data on user behavior, product details, transactions, and much more. The traditional approach to handling large data volumes is to distribute the data across multiple nodes or servers [46]. However, distributing high-dimensional vector data is not straightforward. Unlike traditional relational data, where each record is independent and can be easily partitioned, high-dimensional vectors often need to be compared with one another during the similarity searches [64], which complicates the distribution process. Moreover, ANN search algorithms are commonly used in vector databases to speed up the search process. Existing algorithms use in-memory indexes for low latency and high throughput. However, as the data scales, these algorithms may still suffer from increased latency since it becomes expensive to load all the date into the memory and disk-based indexes must be considered. Furthermore, the infrastructure supporting the vector database will play a significant role in query performance. Techniques such as data sharding, load balancing, and efficient use of hardware will be crucial in ensuring that the system can handle high-concurrency, low-latency queries. DiskANN algorithms [25, 54] index 5-10 times more points/machine using inexpensive SSDs with less than 10ms latency, which is close to the query latency in the in-memory indexes. As we move into a future characterized by increasingly large volumes of high-dimensional data, research and development in this area will be crucial. Both data distribution strategies and query optimization techniques will need to evolve in tandem with the growing demands placed on these systems.

4.5 Data Privacy and Security

In the era of big data and machine learning, the issue of data privacy and security has taken center stage [49, 63]. Like any database system, a vector database is potentially vulnerable to unauthorized access and data breaches. As vector databases become more widely used and the data they store becomes increasingly valuable, they will inevitably become attractive targets for cyber-attacks [51]. This necessitates robust security measures to prevent unauthorized access. Security protocols should include strong access controls, secure data transmission, and encryption strategies to protect the stored data. Moreover, constant monitoring and audit trails can help detect any suspicious activities. Implementing these measures requires a deep understanding of both database security and the specific vulnerabilities that may be unique to vector databases. Beyond the conventional concerns of data security, vector databases present another dimension of privacy challenge: the potential sensitivity of the vector representations themselves. High-dimensional vectors in these databases often capture meaningful patterns and structures in the data they represent. For instance, a vector could represent a user's behavior patterns, a patient's medical record, or a document's semantic content. This raises significant privacy concerns, as individuals might be identifiable from their corresponding vectors, even when the original data is anonymized. A related concern is the potential for 'information leakage' from the vector representations . In some scenarios, it might be possible to reverse-engineer sensitive information from the vectors, especially if the method used to generate the vectors is known [45]. This requires careful consideration of how vectors are generated and how they can be sufficiently anonymized or perturbed to prevent such leakage.

4.6 Hardware Efficiency

The growing dependence on vector databases for high-dimensional data processing has highlighted the importance of hardware efficiency. These databases often utilize approximate nearest neighbor (ANN) search algorithms to find similar vectors, resulting in a significant computational load. This challenge is further amplified in distributed systems that rely on GPUs, DRAM, and other hardware components, where fully utilizing hardware capabilities is crucial for performance but often difficult to achieve. Recently, GPU-based ANN indexes such as SONG [66] and GGNN [50] have been proposed, achieving a two orders of magnitude speedup compared to CPU-based methods for ANN search. However, there are two factors to consider when delegating distance computation to GPUs. First, GPUs require interaction with the host's software and/or hardware layers, resulting in data transfer overhead for computation. Second, distance computations can be performed using a few simple, lightweight vector processing units, making GPUs a less cost-efficient choice for these tasks. In addition, a software-hardware collaborative approach can combine software and hardware components to achieve highly scalable approximate nearest neighbor search services. A study called CXL-ANNS [52] disaggregates DRAM from the host via Compute Express Link (CXL) and places all essential datasets into its memory pool. Another study, FANNS [55], automatically co-designs hardware and algorithms on FPGAs when given a user-provided recall requirement on a dataset and a hardware resource budget. Compared to purely CPU- or GPU-based methods, these software-hardware collaborative approaches achieve superior performance.

5 Conclusions

As LLMs continue to be deployed in various domains, the demand for ANN search on vector representations is expected to surge exponentially. The current vector search libraries and database products still fall short of offering the comprehensive services on vector data. This underscores a significant gap and emphasizes the vast scope of work that remains to be addressed. In the near future, two main challenges may stand out: managing the scalability in the context of an exponentially increasing volume of vectors, and harnessing the potential of embedding models to enhance indexing and querying process. These challenges will likely be key areas of research moving forward.

6 Acknowledgments

This work was partially conducted in the JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust, and was supported in part by NSFC (Grant No. 62372194), National Key Research and Development Program of China under Grant No. 2021YFC3300303, Hubei Natural Science Foundation (Grant No. 2020CFB871), and Zilliz.

References

[1] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. STOC, 1998.

[2] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. VLDB, 1999.

[3] A. Andoni and P. Indyk. LSH algorithm and implementation (E2LSH). `https://www.mit.edu/~andoni/LSH`, 2016.

[4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. Symposium on Computational Geometry, 2004.

[5] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. SIGMOD, 2009.

[6] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. VLDB, 2007.

[7] Y. Lei, Q. Huang, M. S. Kankanhalli, and A. K. H. Tung. Locality-Sensitive Hashing Scheme based on Longest Circular Co-Substring. SIGMOD, 2020.

[8] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. SIGMOD, 2012.

[9] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor. PVLDB, 2015.

[10] K. Lu and M. Kudo. R2LSH: A Nearest Neighbor Search Scheme Based on Two-dimensional. ICDE, 2020.

[11] K. Lu, H. Wang, W. Wang, and M. Kudo. VHP: Approximate Nearest Neighbor Search via Virtual Hypersphere. VLDB, 2020.

[12] W. Liu, H. Wang, Y. Zhang, W. Wang, and L. Qin. I-LSH: I/O Efficient c-Approximate Nearest Neighbor Search in High-Dimensional Space. ICDE, 2019.

[13] W. Liu, H. Wang, Y. Zhang, W. Wang, L. Qin, and X. Lin. EI-LSH: An early-termination driven I/O efficient incremental c-approximate nearest neighbor search. VLDB, 2021.

[14] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin. SRS: Solving c-Approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. PVLDB, 2014.

[15] B. Zheng, X. Zhao, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. Proc. VLDB Endow, 2020.

[16] A. Andoni and P. Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. FOCS, 2006.

[17] A. Andoni and I. P. Razenshtey. Optimal Data-Dependent Hashing for Approximate Near Neighbors. STOC, 2015.

[18] M. Li, Y. Wang, P. Zhang, H. Wang, L. Fan, E. Li, and W. Wang. Deep Learning for Approximate Nearest Neighbour Search: A Survey and Future Directions. TKDE, 2023.

[19] Y. Tian, X. Zhao, and X. Zhou. DB-LSH: Locality-Sensitive Hashing with Query-based Dynamic Bucketing. ICDE, 2022.

[20] Y. Tian, X. Zhao, and X. Zhou. DB-LSH 2.0: Locality-Sensitive Hashing with Query-based Dynamic Bucketing. TKDE, 2023.

[21] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. WWW, 2005.

[22] X. Zhao, B. Zheng, X. Yi, X. Luan, C. Xie, X. Zhou, and C. S. Jensen. FARGO: Fast Maximum Inner Product Search via Global Multi-Probing. PVLDB, 2023.

[23] Q. Huang, Y. Lei, and A. K. H. Tung. Point-to-Hyperplane Nearest Neighbor Search Beyond the Unit Hypersphere. SIGMOD, 2021.

[24] ANN Benchmarks. https://ann-benchmarks.com/.

[25] S. Jaiswal, R. Krishnaswamy, A. Garg, H. V. Simhadri, and S. Agrawal. OOD-DiskANN: Efficient and Scalable Graph ANNS for Out-of-Distribution Queries. CoRR, 2022.

[26] S. Gollapudi, N. Kari, V. Sivashankar, R. Krishnaswamy, N. Begwani, S. Raz, Y. Lin, Y. Zhang, N. Mahapatro, P. Srinivasan, A. Singh, and H. V. Simhadri. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. WWW, 2023.

[27] 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. PVLDB, 2020.

[28] Q. Zhang, S. Xu, Q. Chen, G. Sui, J. Xie, Z. Cai, Y. Chen, Y. He, Y. Yang, F. Yang, M. Yang, and L. Zhou. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. OSDI, 2023.

[29] A. Babenko and V. S. Lempitsky. The inverted multi-index. IEEE Trans. Pattern Anal. Mach. Intell., 37(6):1247–1260, 2015.

[30] A. Babenko and V. S. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In CVPR, pages 2055–2063. IEEE Computer Society, 2016.

[31] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. Proc. VLDB Endow., 12(2):112–127, 2018.

[32] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. Proc. VLDB Endow., 13(3):403–420, 2019.

[33] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In CVPR, pages 2946–2953. IEEE Computer Society, 2013.

[34] J.-P. Heo, Z. Lin, and S.-E. Yoon. Distance encoded product quantization for approximate k-nearest neighbor search in high-dimensional space. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(9):2084–2097, 2019.

[35] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell., 33(1):117–128, 2011.

[36] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In CVPR, pages 2329–2336. IEEE Computer Society, 2014.

[37] C. Li, M. Zhang, D. G. Andersen, and Y. He. Improving approximate nearest neighbor search through learned adaptive early termination. In SIGMOD Conference, pages 2539–2554. ACM, 2020.

[38] H. Noh, T. Kim, and J.-P. Heo. Product quantizer aware inverted index for scalable nearest neighbor search. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 12210–12218, October 2021.

[39] J. Paparrizos and L. Gravano. k-shape: Efficient and accurate clustering of time series. In SIGMOD Conference, pages 1855–1870. ACM, 2015.

[40] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In KDD, pages 262–270. ACM, 2012.

[41] B. Zheng, Z. Yue, Q. Hu, X. Yi, X. Luan, C. Xie, X. Zhou, and C. S. Jensen. Learned probing cardinality estimation for high-dimensional approximate NN search. In ICDE, pages 3209–3221. IEEE, 2023.

[42] Faiss. https://github.com/facebookresearch/faiss

[43] NMSLib. https://github.com/nmslib/nmslib

[44] B. Bratic, M. E. Houle, V. Kurbalija, V. Oria, and M. Radovanovic. Nn-descent on high-dimensional data. In WIMS, pages 20:1–20:8, 2018.

[45] S. Chen, H. Khanpour, C. Liu, and W. Yang. Learning to reverse dnns from AI programs automatically. CoRR, abs/2205.10364, 2022.

[46] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik. Scalable distributed stream processing. In CIDR, 2003.

[47] C. Fu and D. Cai. EFANNA : An extremely fast approximate nearest neighbor search algorithm based on knn graph. CoRR, abs/1609.07228, 2016.

[48] C. Fu, C. Xiang, C. Wang, and D. Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. PVLDB, 12(5):461–474, 2019.

[49] Z. Gheid and Y. Challal. Efficient and privacy-preserving k-means clustering for big data mining. In Trustcom/BigDataSE/ISPA, pages 791–798. IEEE, 2016.

[50] F. Groh, L. Ruppert, P. Wieschollek, and H. P. A. Lensch. GGNN: graph-based GPU nearest neighbor search. IEEE Trans. Big Data, 9(1):267–279, 2023.

[51] E. Irmak and I. Erkek. An overview of cyber-attack vectors on scada systems. In 2018 6th International Symposium on Digital Forensic and Security (ISDFS), pages 1–5, 2018.

[52] J. Jang, H. Choi, H. Bae, S. Lee, M. Kwon, and M. Jung. CXL-ANNS: software-hardware collaborative memory disaggregation and computation for billion-scale approximate nearest neighbor search. In USENIX Annual Technical Conference, pages 585–600. USENIX Association, 2023.

[53] J. W. Jaromczyk and M. Kowaluk. Constructing the relative neighborhood graph in 3-dimensional euclidean space. Discret. Appl. Math., 31(2):181–191, 1991.

[54] 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. In Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019.

[55] W. Jiang, S. Li, Y. Zhu, J. de Fine Licht, Z. He, R. Shi, C. Renggli, S. Zhang, T. Rekatsinas, T. Hoefler, and G. Alonso. Co-design hardware and algorithm for vector search. CoRR, abs/2306.11182, 2023.

[56] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. IEEE Trans. Big Data, 7(3):535–547, 2021.

[57] P. Lin and W. Zhao. On the merge of k-nn graph. CoRR, abs/1908.00814, 2019.

[58] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell., 42(4):824–836, 2020.

[59] J. A. V. Muñoz, M. A. Gonçalves, Z. Dias, and R. da Silva Torres. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. Pattern Recognit., 96, 2019.

[60] A. Singh, S. J. Subramanya, R. Krishnaswamy, and H. V. Simhadri. Freshdiskann: A fast and accurate graph-based ANN index for streaming similarity search. CoRR, abs/2105.09613, 2021.

[61] B. I. Tingle, K. G. Tang, M. Castanon, J. J. Gutierrez, M. Khurelbaatar, C. Dandarchuluun, Y. S. Moroz, and J. J. Irwin. Zinc-22–a free multi-billion-scale database of tangible compounds for ligand discovery. J. Chem. Inf. Model., 63(4):1166–1176, 2023.

[62] M. Wang, X. Xu, Q. Yue, and Y. Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. PVLDB, 14(11):1964–1978, 2021.

[63] R. Wang, Y. F. Li, X. Wang, H. Tang, and X. Zhou. Learning your identity and disease from research papers: information leaks in genome wide association study. In CCS, pages 534–544. ACM, 2009.

[64] M. Yang, Y. Zuo, M. Chen, and X. Yu. Scalable distributed knn processing on clustered data streams. IEEE Access, 7:103198–103208, 2019.

[65] W. Zhao. k-nn graph construction: a generic online approach. CoRR, abs/1804.03032, 2018.

[66] W. Zhao, S. Tan, and P. Li. SONG: approximate nearest neighbor search on GPU. In ICDE, pages 1033–1044. IEEE, 2020.

[67] X. Zhao, Y. Tian, K. Huang, B. Zheng, and X. Zhou. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. Proc. VLDB Endow., 16(8):1979–1991, 2023.

[68] Bernhardsson, E. Annoy: Approximate Nearest Neighbors in C++/Python. https://pypi.org/project/annoy/, Python package version 1.13.0, 2018

[69] Malkov, Y. & Yashunin, D. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Trans. Pattern Anal. Mach. Intell. 42, 824-836, 2020

[70] Sun, Y., Wang, W., Qin, J., Zhang, Y. & Lin, X. SRS: Solving c-Approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. Proc. VLDB Endow.. **8**, 1-12, 2014

[71] Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W. & Lin, X. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. IEEE Transactions On Knowledge And Data Engineering. **32**, 1475-1488, 2020

[72] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., et al. Language Models are Few-Shot Learners. NeurIPS. (2020)

[73] OpenAI GPT-4 Technical Report. CoRR. abs/2303.08774 (2023)

[74] Patel, A., Li, B., Rasooli, M., Constant, N., Raffel, C. & Callison-Burch, C. Bidirectional Language Models Are Also

Few-shot Learners. ICLR. (2023)
[75] Pinecone. https://docs.pinecone.io
[76] Milvus. https://milvus.io/
[77] Weaviate. https://weaviate.io/
[78] Azizi, I., Echihabi, K. & Palpanas, T. ELPIS: Graph-Based Similarity Search for Scalable Data Science. Proceedings Of The VLDB Endowment. **16**, 1548-1559 (2023)
[79] Echihabi, K., Fatourou, P., Zoumpatianos, K., Palpanas, T. & Benbrahim, H. Hercules Against Data Series Similarity Search. Proc. VLDB Endow.. **15**, 2005-2018 (2022)
[80] Chatzakis, M., Fatourou, P., Kosmas, E., Palpanas, T. & Peng, B. Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. Proc. VLDB Endow.. **16**, 1140-1153 (2023)