Bulletin of the Technical Community on

Data Engineering

September 2023 Vol. 47 No. 3 IEEE Computer Society

Letters

Special Issue on High-Dimensional Similarity Search: from Time Series Management Systems to Vector Databases

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Community on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

This special issue of the IEEE Data Engineering Bulletin is dedicated to a timely subject: high-dimensional similarity searches.

High-dimensional similarity searches are indispensable in a variety of fields. For instance, they are employed for time series analysis and forecasting, which have numerous applications in science, medicine, and business. Recently, the subject of vector databases has garnered significant attention, primarily driven by the emergence of Large Language Models (LLMs) and Retrieval Augmented Generation (RAG), where vector databases have assumed a crucial role in facilitating various applications in the domain of generative AI ranging from chatbots to AI agents. On the other hand, the continuous progress and widespread adoption of vector databases have been occurring over the past few decades. Specifically, it brought about a revolution in the domain of information retrieval by enhancing the traditional approach of term-based retrieval with embedding-based retrieval. This novel technique, which involves utilizing vector-based semantic search, significantly enhances the ability to retrieve relevant information.

This special issue, curated by Associate Editor Themis Palpanas, gathers insights from various branches of this field. The articles contained within this special issue provide an array of theoretical and practical solutions for similarity searches in high-dimensional spaces. Our authors investigate algorithms to improve similarity searches, explore the evolution of graph-and tree-based indexes, and chart the course for improving data management systems. The articles delve into aspects of optimal design strategies to enhance computational efficiency in the era of AI, examine the deployment of techniques like locality-sensitive hashing and product quantization, and delve into the improvements dynamic space partitions can offer for faster and more accurate searches.

We believe the topic of vector databases is significant for the database community, for its role as a bridge connecting data management and artificial intelligence. While considerable progress has been made in the field of vector-based similarity search, it is crucial to recognize the existence of ongoing obstacles that require attention and resolution. For example, there exist numerous relationships among the vast amount of data represented by the vector database. Given a question, how do we ensure all pertinent information is retrieved through vector search? Additionally, what strategies can be employed to narrow the disparity between pre-training an LLM on the vast amount of data and in-context learning via RAG, which disregards the majority of the data that could potentially have relevance to the given question? Thus, our shared objective is to enhance the advancement of this discipline, and our overarching aspiration encompasses two expansive domains: database and artificial intelligence.

We would like to express our profound gratitude to all the authors who contributed to this issue, to Themis Palpanas for bringing these insightful articles to the forefront, and to Nurendra Choudhary for his assistance in the publication process.

Haixun Wang
Instacart

Letter from the Special Issue Editor

Similarity search in high-dimensional data spaces was a relevant and challenging data management problem in the early 1970s, when the first solutions to this problem were proposed. Today, fifty years later, we can safely say that the exact same problem is more relevant (from Time Series Management Systems to Vector Databases) and challenging than ever. This is true, not because the research community has been idle; on the contrary, the literature on this topic is very large and diverse, demonstrating both the interest in this problem, as well as the wide range of ideas that have been applied to it and led to impressive advances. This is true, rather because very large amounts of high-dimensional data are now omnipresent (ranging from traditional multidimensional data to time series and deep embeddings), and the performance requirements (i.e., response-time and accuracy) of a variety of applications that need to process and analyze these data have become very stringent and demanding.

In these past fifty years, high-dimensional similarity search has been studied in its many flavors. Similarity search algorithms for exact and approximate, one-off and progressive query answering. Approximate algorithms with and without (deterministic or probabilistic) quality guarantees. Solutions for on-disk and in-memory data, static and streaming data. Approaches based on multidimensional space-partitioning and metric trees, random projections and locality-sensitive hashing (LSH), product quantization (PQ) and inverted files, k-nearest neighbor graphs and optimized linear scans.

Another interesting aspect of the work in high-dimensional similarity search is that research on this problem has been conducted by different (sub-)communities in a somewhat independent fashion, that is, with not much interaction among them. A notable example is the work on data-series (and time-series) similarity search, which was recently shown to achieve the state-of-the-art performance for several variations of the problem, on both time-series and general high-dimensional vector data. It is only very recently that a conscious effort is being made in order to gather the state-of-the-art methods from these different communities, and thus, enable the comparison of the various approaches, the extraction of useful insights, and the development of improved solutions. This special issue contributes to this effort by including a selection of papers that represent the research activity in several of these communities, highlighting similarities and differences, discussing the results of some initial cross-pollination (that has already started taking place), and revealing open research directions.

In the first paper, Wang et al. summarize and discuss state-of-the-art solutions for approximate similarity search based on k-nearest neighbor graphs, data-series tree indexes, as well as their combination, and point to promising research directions. In the second paper, Zhang et al. list the similarity search requirements of modern applications, and present novel algorithms based on k-nearest neighbor graphs that exploit multi-core architectures and NVMe memory. In the third paper, Tian et al. summarize and discuss state-of-the-art solutions, as well as future research directions, for approximate similarity search based on locality-sensitive hashing, product quantization and k-nearest neighbor graphs, as well as combinations of these methods. In the fourth paper, Dong et al. propose the idea of learning high-quality space partitions, and develop a novel solution that combines k-nearest neighbor graphs with supervised learning (including deep neural networks) for approximate similarity search. In the fifth paper, Paparrizos et al. compare many distance measures proposed for time-series similarity search, comment on the lower bounds that speedup some of these measures, and discuss the open research problems that their findings point to. Finally, in the sixth paper, Aumüller and Ceccarello study the very important problem of creating appropriate benchmarks for approximate similarity search; they review recent benchmarks, and offer guidelines for future efforts in this area.

Overall, the above papers represent an interesting sample of the ongoing work on high-dimensional similarity search. We hope that this special issue will further help and inspire the research community in its quest to solve this challenging problem. We would like to thank all the authors for their valuable contributions, as well as Haixun Wang for giving us the opportunity to put together this special issue, and Nurendra Choudhary for his help in its publication.

Themis Palpanas
Université Paris Cité

Graph- and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions

Zeyu Wang[†], Peng Wang[†*], Themis Palpanas[‡], Wei Wang[†]
[†]Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
{wangzeyu17, pengwang5, weiwang1}@fudan.edu.cn
[‡]LIPADE, Université Paris Cité & IUF, themis@mi.parisdescartes.fr

Abstract

Approximate nearest neighbor search on high-dimensional vectors is a crucial component for numerous applications in various fields. To solve this problem efficiently, dozens of indexes have been proposed in the past decades. Among them, graph-based indexes show superior query performance in memory while tree-based indexes achieve the best scalability and building efficiency. In this paper, we systematically study the evolution of these two kinds of indexes and the recent progress with ablation studies and analyses, which help understand where the performance improvement comes from and the difference between different index families. Moreover, we conduct a comparative study over these two index families and discuss the existing and potential combinations of them. We believe this study can serve as a guide to the most promising directions for addressing the open problems in this area.

1 Introduction

Approximate nearest neighbor search (ANNS) on high-dimensional vectors is a crucial component for numerous applications in various fields, such as web search engines [15], image retrieval [72], recommendation systems [18], and large language models [1]. Recent studies have further shown that deep neural networks can be augmented by retrieval to enhance accuracy on the classical problems [40] (e.g., open-domain query-answering problem [14]) and decrease the magnitude of parameters [33], further emphasizing the significance of ANNS in modern AI applications. Objects, such as images, documents, and videos, can be transformed into dense vectors in the embedding space. Given a query vector $q \in \mathbb{R}^D$ and a distance measure $dist(\cdot, \cdot)$, ANNS aims to find top-k most similar objects (i.e., $kNN(q)$) in the embedding space $\mathbb{R}^D$ of the dataset. Since the cost to find the exact $kNN(q)$ is prohibitively high, ANNS finds the approximate nearest neighbors $A(q)$ instead. The search accuracy is measured by recall, defined as $Recall@k = \frac{1}{|Q|}\sum_{q\in Q}\frac{|A(q)\cap kNN(q)|}{k}$. For many advanced ANNS algorithms, the recall can reach over 99% with hundreds to thousands of times of speedup over the linear scan. Therefore, they quickly become practical and well-recognized solutions for the applications mentioned above.

A common way of ANNS algorithms is to first build an index for the dataset and probe the index when processing queries. In the past decades, researchers have designed dozens of index structures to solve ANNS problems. They can be roughly categorized into four index families: graph-based [69], tree-based [24, 51], quantization-based [50] and Locality Sensitive Hashing (LSH)-based [8, 83] index family (other promising variations of these ideas have also been proposed [38, 39]). Although all these indexes have their specialties and advantages in certain scenarios, we focus on graph-based and tree-based indexes in this paper, because of their wide use on production systems [2, 66, 72], and the promising future of the combination of these two techniques.

*Corresponding author

Graph-based indexes represent the dataset with a graph where each point is one vector. The query processing algorithm starts from some entry point(s) and finds the approximate nearest neighbors by stepping towards the query along the edges of the graph. With an appropriate graph structure, the search route can converge to the neighbors of the query in only a few steps, which is called the navigability of graph-based indexes. As extensively evaluated, graph-based indexes are the most efficient and accurate in-memory ANNS index when querying [69].

Tree-based indexes on high-dimensional data are widely adopted by the industry (e.g., Postgres [2], SPTAG [16], ADB-V [72]) and famous open-source libraries (e.g., FAISS [36], Scikit-learn [4]), because of its outstanding scalability, robust performance, index-building efficiency, and high interpretability. Classical tree-based indexes like kd-tree [61], and ball-tree [12] suffer from the curse of dimensionality. In high-dimensional space, simple partitioning cannot effectively cluster similar vectors. In recent years, many classical tree indexes have been re-designed to adapt to high-dimensional space. These indexes show the advantage on large-scale datasets for in-memory and disk indexes, stand-alone and distributed environments, as well as accuracy-guaranteed and exact kNN search [15, 23, 24].

Although the design rationales of graph- and tree-based indexes are vastly different, we observe that there are some common design philosophies, as well as some techniques that could be complementary to each other. As verified by results of recent studies, solving the scalability problem of graph-based index turns tractable by combining the strengths of these two kinds of indexes. Moreover, we further propose four important open problems and directions that could be solved by this combination.

Our paper is organized as follows. In Section 2, we review the structure evolution of the graph-based index and analyzes its performance by ablation studies. In Section 3, we summarize the recent progress of tree-based indexes and discuss their relationships. In Section 4, we conduct a comparative study of the two kinds of indexes, and discuss open research problems. We conclude this paper in Section 5.

2 Graph-based Index Family

Graph-based index builds a directed, unweighted graph $G = (V, E)$ as the index where each vertex [1] $v \in V$ represents a vector in the dataset and the directed edge in E represents some kind of proximity between two vectors. The widely-used greedy search algorithm starts from one or a group of entry points and approaches the query step by step. Two priority queues C and Res are maintained in this process, ordered by the distance to the query. C (with unlimited size) stores the points to be visited, while Res stores only the ef nearest points that have been checked before. In each step, the first point from C is popped and all its out-neighbors will be checked. If some neighbor point's distance to the query is smaller than the furthest point in Res, it will be pushed into C and Res. The algorithm terminates when the distance of the first point in C is larger than all the ones in Res. Finally, the closest k elements in Res will be returned. Apparently, ef is an important parameter which controls the accuracy-efficiency trade-off. The query processing algorithm consists of two stages [68, 73], a fast navigating stage leading the search route from the entry point(s) to the query's neighbors, and a recall stage that traverses the neighbors of the query to obtain the final kNN results. Several extensions and improvements of this search algorithm have recently been proposed, employing query parallelization [58], GPUs [75, 82], learned early termination [42], and guided search [73]. In this section, we study the basic, common search algorithm, and focus on the structure of the graph index.

2.1 Overview

In this subsection, we briefly review the building process of graph-based indexes, as well as the structures they employ. The design inheritance is shown in Figure 1.

[1] In this paper, we use vertex, node, point, and vector for graph-based indexes interchangeably.

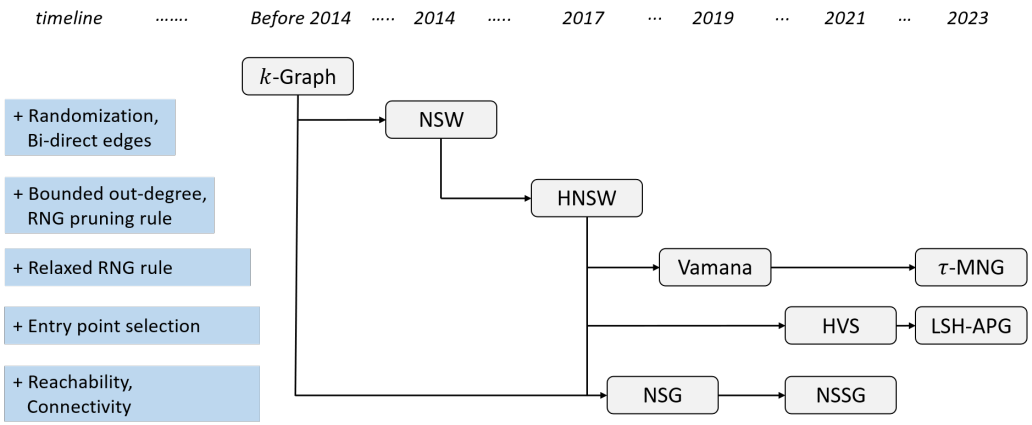


Figure 1: The evolution of the structures of in-memory graph-based indexes. The solid arrows denote the inheritance of designs.

K-Graph [19] Each node in K-Graph out-links its K nearest neighbors in the graph. To reduce the linear complexity of index building, a dozen of accelerating techniques are proposed [19, 28].

Navigable Small World Graph (NSW) [49] NSW is built by sequential insertions of points of the dataset. When inserting a point v, we find M nearest neighbors on the previously-inserted sub-dataset by the greedy search algorithm. Then NSW adds bi-directional links between v and the M nearest neighbors. Note that NSW does not have a maximal out-degree limit, and the NSW index built with parameter M has nearly the same number of edges as the K-Graph where $K = 2M$.

Hierarchical Navigable Small World Graph (HNSW) [48] HNSW has a hierarchical structure where the graph in the lower layer contains all the points in the upper layers and the bottom layer HNSW_0 contains all points in the dataset. The number of points is reduced exponentially bottom-up, and the level of a point is decided randomly. The query starts from the top layer to find NNs as the entry points of the next layer until reaching the bottom layer. Like NSW, HNSW is also built by sequential insertion and greedy search, but with a limited out-degree $2M$. Once a point's out-degree exceeds $2M$, HNSW prunes neighbors using the RNG rule: this is derived from the Relative Neighborhood Graph [65]. Simply speaking, the RNG rule removes the longest edge in the triangles existing in the graph. In this way, HNSW tries to preserve the reachability of the graph index while limiting the out-degree, by removing the "redundant" edges, at the expense of a few more hops when querying.

Relaxed pruning rule: Vamana [64] and τ-MNG [55] Since the heuristic RNG pruning rule tends to be "strict", recent studies have verified that some useful edges are also pruned, which leads to a sub-optimal graph structure. Vamana relaxes the RNG rule by allowing the inclusion of the longest edge in a triangle, if it is not $1 + \alpha$ times longer than the second longest edge, where α is a user-defined relaxing parameter. In this way, Vamana shortens the search path when querying, which helps reduce random I/Os when the index is on disk. A recent study [55] points out that RNG rule is ineffective when the query does not belong to the dataset, and mitigates this problem by relaxing the RNG pruning rule with a parameter τ. We omit the details for lack of space. The resulting approach, τ-MNG, is expected to have a lower search time complexity than other graph-based indexes when the distance between the query and its nearest neighbor is smaller than τ.

Better entry points: HVS [47] and LSH-APG [81] HVS and LSH-APG use auxiliary data structures to obtain better entry points for the graph-based indexes. HVS leverages an adaptive hierarchical tree index with

quantization techniques, while LSH-APG builds multiple LSB-trees for coarse-grained indexing. The graph-based index is HNSW_0 [47], or even a simpler variant [81].

Better reachability: NSG [29] and NSSG [30] Although HNSW achieves prominent average query performance, it cannot guarantee the reachability to any point in the graph index. Since accomplishing the reachability starting from any point in the graph is intractable, recent graph-based indexes opt to fix the entry point(s) and guarantee the reachability from these entry points. NSG uses the medoid point of the dataset as the entry point, and refines K-Graph to ensure reachability. Specifically, for a specific point v, NSG regards v as a query and performs approximate greedy search in the graph index. The visited points during the search (along with the neighbors of v in K-Graph) become the candidate neighbors of v, and are then pruned by RNG rule to limit the out-degree. After iterating all the points, NSG tests the reachability with a Deep-First-Search Tree rooted at the entry point. The isolated points will be linked by the nearest point in the tree. In contrast to NSG, NSSG randomly selects a group of points as entry points, and only uses two-hop neighbors of a point in K-Graph as the candidates of neighbors for pruning. Besides the reachability to the point in the dataset, NSSG also considers the points that do not belong to the dataset, by studying the trade-off between the navigability and the sparsity of the graph index under the RNG rule.

2.2 Understanding the Evolution with Ablation Studies

Although the heuristics of advanced graph-based indexes have been stated in the papers, it is yet not clear why the proposed techniques work well. In this subsection, we describe the evolutionary process from K-Graph to NSW, HNSW and more optimizations on HNSW by abundant ablation studies. We aim to help readers understand which techniques actually enhance the performance and the reasons behind their effectiveness from a novel perspective, which can inspire the future designs of ANNS algorithms.

2.2.1 The problem of K-Graph

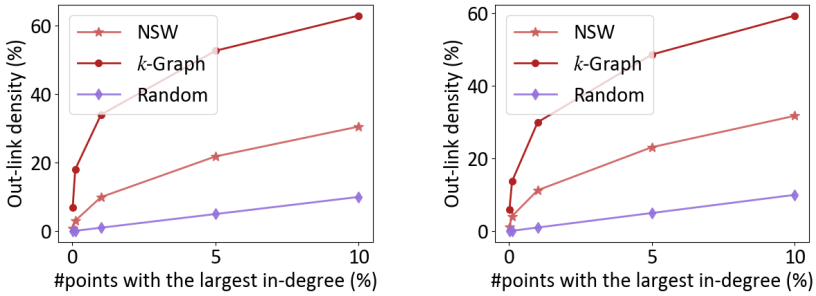
We start by examining the K-Graph approach, because not only K-Graph is a graph-based index, but it also exhibits some inherent properties (detailed below) of high-dimensional vectors in the context of ANNS problem. In this paper, we use exact K-Graphs instead of approximate ones to avoid the effect of approximation errors. A common argument in the literature for the unsatisfactory query performance of K-Graph is the loss of navigability [48, 69], which is verified on real datasets in [34]. However, it is not yet clear the root cause of the loss of navigability. According to previous studies, the distribution of in-degree of points is very skewed [59]. That is, a small portion of points occupy most of the kNN lists for all the points while most of the other points are rarely linked by others. This is also verified by our experiments on real datasets as shown in Table 1 with skewness (standardized third moment). For simplicity, we denote the in-degree of a point in K-Graph by k-occurrence [59]. Note that k-occurrence is a property of a point in a dataset, independent of the type of index built on the dataset.

Table 1: Skewness of in-degree distribution of graph-based indexes

	Deep1M	SIFT1M
K-Graph	2.436	1.875
NSW	4.642	4.107
pNSW	4.025	3.457
HNSW_0	1.209	1.395

Furthermore, we observe that in K-Graph the points with high in-degree (i.e., hubs) always interconnect with each other. To describe this phenomenon, we define the out-link density of a group of points $X \subset V$ as $\rho_{out-link}(X) = \frac{|\{(x,y)|x,y\in X \wedge (x,y)\in E\}|}{|\{(x,y)|x\in X \wedge y\in V \wedge (x,y)\in E\}|}$.The out-link density describes how much a group of points is interconnected. A high out-link density means these points tend to connect to themselves rather than other points in the graph. In Figure 2, we select different number of hubs and compute the average out-link density of these hubs. The baseline is a random graph, where each point has K out-neighbors selected uniformly

at random. As we can see, the out-link density of the hubs in K-Graph is much higher than the random graph and NSW. Over half of the out-links of the 5% hubs connect to themselves instead of other points.



(a) Deep1M (b) Sift1M

Figure 2: Out-link density of the group of points with the largest in-degree.

This phenomenon leads to local optima in K-Graph, which is a classical problem of the greedy search algorithm. Consider an extreme case where $ef = 1$ for querying, i.e., Res only contains the nearest checked point to the query, which means that the points further than the current visited point will never be visited. In other words, no backtracking exists in the search route. In this case, if the search route steps into a point which is closer to the query than all its neighbors, the search algorithm terminates directly, as shown in Figure 3, and the actual nearest neighbors (that could be reached by detouring) are missed. The termination point is a local optimum, in contrast to the global optimum (i.e., real kNN). Since in K-Graph, most of the out-neighbors of a hub are also hubs, once the search route steps into one hub, it is hard to escape from the local optimum to other regions, which may actually contain the true nearest neighbors. Worse still, the hubs are frequently visited when querying since the points with high in-degree have a higher probability to be visited when starting from a random point. As a result, K-Graph shows weak navigability to acquire real kNN even with a large ef.

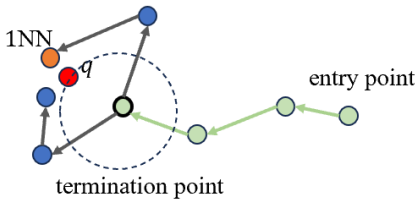


Figure 3: An example of local optimum in the greedy search algorithm.

We further verify this impact by analyzing the returned results of the greedy search algorithms. Formally, given a query q, we collect the true positive set $S_{TP} = A(q) \cap kNN(q)$, the false positive set $S_{FP} = A(q) - kNN(q)$ and the false negative set $S_{FN} = kNN(q) - A(q)$, and compute the average k-occurrence of these point sets, denoted by $k^{occur}_{S_{TP}}$, $k^{occur}_{S_{FP}}$ and $k^{occur}_{S_{FN}}$, respectively. In the experiments, we use the same query algorithm and the same sparsity of the graphs with K=32 for K-Graph and M=16 for the other three. As shown in Table 2, for K-Graph, the difference between $k^{occur}_{S_{TP}}$ and $k^{occur}_{S_{FN}}$ is the largest, indicating that the points with higher in-degree are easier to be recalled whereas the missing points are markedly less connected. Moreover, the average k-occurrence of points in S_{FP} is also higher than S_{FN} for K-Graph, and to achieve a higher recall (95%), K-Graph needs the most number of steps compared with other indexes. These results verify the above inference that the search route in K-Graph is prone to get stuck in the local optimum formed by the hubs and thus needs to pay more effort to escape from this trap.

Recall that the problems of K-Graph is an intrinsic problem for all graph-based indexes on high-dimensional data. On the one hand, the local proximity of K-Graph is crucial in the recall stage during the search process [68]. On the other, over-rich local connections in the graph lead to more local optimum which hinders the search accuracy.

Table 2: Average k-occurrence of different sets returned by the greedy search algorithm on different graph-based indexes. #hops are the average number of points visited in the search route. NDC stands for the average Number of Distance Calculations during search, which represents the time cost.

Dataset	Accuracy	Index	$k^{occur}_{S_{TP}}$	$k^{occur}_{S_{FP}}$	$k^{occur}_{S_{FN}}$	$k^{occur}_{S_{TP}} - k^{occur}_{S_{FN}}$	#hops	NDC
Deep1M	recall@50=0.90	K-Graph	61.08	41.86	14.44	46.64	116	1716
		NSW	61.06	40.72	32.99	28.07	**57**	1709
		pNSW	62.15	46.36	24.66	37.49	90	1379
		$HNSW_0$	61.00	45.46	37.05	**23.95**	79	**1287**
	recall@50=0.95	K-Graph	60.38	27.30	7.62	52.76	176	2319
		NSW	60.17	28.28	19.55	40.62	**108**	2734
		pNSW	61.22	31.96	15.37	45.85	162	2170
		$HNSW_0$	60.72	32.57	24.39	**36.33**	**108**	**1781**
SIFT1M	recall@50=0.90	K-Graph	53.81	28.90	11.72	42.09	386	4927
		NSW	54.07	41.78	37.86	16.21	**70**	2015
		pNSW	54.98	45.44	31.84	23.14	106	1690
		$HNSW_0$	54.18	44.40	40.10	**14.08**	71	**1336**
	recall@50=0.95	K-Graph	52.86	37.27	17.17	35.69	423	5336
		NSW	54.03	32.77	27.71	26.32	112	2881
		pNSW	54.55	35.59	23.01	31.54	169	2464
		$HNSW_0$	54.03	34.04	29.70	**24.33**	**108**	**1890**

2.2.2 The Benefit of Long-range Connections

To mitigate the problem of K-Graph, NSW introduces long-range connections to break the traps formed by hubs, and enhance global navigability. The long-range connections can be defined as the edges that do not exist in the K-Graph (whose graph sparsity is similar to NSW). NSW introduces these connections by randomization. Recall that when inserting a point, NSW finds its nearest neighbors by greedy search on the existing graph, and then builds bi-directional links with these neighbors. In this way, the degree [2] of points in NSW is significantly influenced by the insertion order: the points inserted earlier have a higher possibility to be linked by other points than the later ones. This is demonstrated in Table 3, where the NSW (second line) Pearson correlation coefficients for k-occurrence and insertion order are (almost) the same. Given that the insertion order is usually random, the lengths of the new links introduced in NSW also tend to be random. Thus, long-range connections are introduced.

We quantify the size of long-range connections by graph quality [34, 69], which is defined as the overlapping ratio of edges with K-Graph under similar sparsity. As shown in Table 5, NSW preserves nearly half of the short edges belonging to K-Graph, while introducing long-range connections for the other half. Although affected by the random selection of the previously inserted points, these connections can substantially improve the navigability of graph indexes, as we explain below. We note that the hubs do not necessarily have high k-occurrence, but are also determined by the insertion order. In other words, the distribution of hubs is "disrupted" by randomization. As a result, the hubs in NSW are not interconnected, and the out-link density of hubs significantly drops compared to K-Graph (see Figure 2). It means that although the hubs are still visited frequently, the cases of local optimum

[2] Note that in NSW, the in-degree of a point is equal to its out-degree since all edges are bi-directional. Thus, we simply use the term degree.

Table 3: Pearson correlation coefficients between in-degree and insertion-order/k-occurrence of points on different indexes.

	Deep1M		Sift1M	
	Ins. ord.	k-occur.	Ins. ord.	k-occur.
K-Graph	0.00	1.00	0.00	1.00
NSW	-0.53	0.56	-0.53	0.56
pNSW	-0.44	0.68	-0.43	0.67
$HNSW_0$	-0.63	0.22	-0.52	0.26

Table 4: Query analyses under recall@1=99%, on different graph-based indexes.

	Deep1M		Sift1M	
	#hops	NDC	#hops	NDC
K-Graph	5001	29315	2047	17698
NSW	137	3244	150	3644
pNSW	156	3074	663	7246
$HNSW_0$	156	2422	111	1943

Table 5: Graph quality of different indexes, where K=32 and M=16

Deep1M	Sift1M
GQ	GQ
100%	100%
51.84%	53.76%
51.82%	52.71%
30.80%	32.17%

are reduced and thus the navigability on NSW is enhanced.

Table 2 shows that in order to achieve the same recall, NSW needs on average only **32**% of the K-Graph steps. The difference between $k^{occur}_{S_{TP}}$ and $k^{occur}_{S_{FN}}$ is reduced by half, clearly demonstrating that the missing nearest neighbors are less susceptible to local optima, and leading to performance improvements. In some cases though, e.g., in the Deep1M dataset, the improvement is marginal and even negative, despite the short search paths. The reason is that the degree of NSW is not fixed. Since the search cost of the greedy search algorithm is the product of the length of the search route and the out-degree of each point in the route, a high out-degree will linearly increase the time complexity. Worse still, the distribution of degree of points in NSW is even more skewed than K-Graph (see Table 1), which further degrades the worst-case time complexity.

A simple optimization is to restrict the out-degree to $2M$ by selecting the nearest $2M$ neighbors. We call this method pruned-NSW (pNSW). pNSW mainly removes the long-range connections, which can be verified by the fact that the graph quality is nearly the same as NSW (see Table 5). As shown in Table 2 pNSW seems have a better performance than NSW by sacrificing a little navigability and achieves a remarkable gain in overall performance. However, as shown in Table 4, under high-recall constraint (99% recall@1), pNSW suffers a significant degradation on Sift1M dataset, nearly two times slower than NSW. The results of pNSW further demonstrate that the long-range connections, especially the links of the hubs, significantly influence the global navigability of the graph index.

2.2.3 The RNG Pruning Rule

Up to this point, there seems to exist a tradeoff in the design of graph indexes between search efficiency and navigability. If we limit the (out-)degree of points for efficiency, navigability will be hurt, as in K-Graph and pNSW. If we break the degree limit, the search efficiency will degrade, as in NSW. To address this situation, we observe that the distribution skewness of the points' in-degree is a key point. A skewed distribution means the existence of hubs in the graph, and given the high visit frequency of these hubs, these hubs bear the responsibility of high navigability. In order to provide navigability, hubs need a high degree and more long-range connections, causing the problem mentioned above. Therefore, if we can render in-degree distribution less skewed, then hubs will no longer exist and the navigability will be provided equally by all the points in the graph. In this case, the bounded degree and the navigability can be achieved at the same time.

HNSW is a nice example of this rationale. Given that the skewness of the k-occurrence distribution is an inherent property of high-dimensional vectors, "amortizing" the navigability to all points is very challenging. HNSW leverages two techniques to accomplish this target. One is randomization, like NSW, which disrupts the distribution of hubs. The other is RNG pruning rule, which is a key design inherited by most of the following graph-based techniques [29, 30, 69]. Specifically, for a point v to be inserted, HNSW first collects sufficient

nearest neighbors on the existing graph as candidates by greedy search, and then prunes these candidate neighbors down to M, using the RNG rule. Finally, like NSW, HNSW tries to add M reversed links from the M selected neighbors to v. Once the reversed link exceeds the out-degree limit $2M$, the RNG rule will be used to prune these neighbors to no more than $2M$. For clarity, we only consider HNSW_0 in this section to avoid the influence of the hierarchy, which will be discussed separately in the next section.

When pruning candidate neighbors, HNSW sorts them according to the distance to the inserted point v and checks them in order. The inserted neighbor c should satisfy that $\forall c' \in C(v), dist(v,c) < dist(c,c')$, where $C(v)$ is the set of already selected neighbor points. Consider the example in Figure 4 (left), where $C = \{c_1, c_2\}$. The next candidate neighbor c will be pruned, because $dist(v,c) > dist(c,c_2)$. The heuristic behind the RNG rule is that for the pruned neighbors like c, even though v does not directly link to c, v can reach c by detouring from c_2. Since c_2 is closer to c, v must not be the local optimum if the query is on, or near to c. As a result, compared to NSW, HNSW filters out many of the M nearest neighbors found during the greedy search, without losing the reachability to them [3]. In this way, the saved slots of out-neighbors can be used to accommodate more long-range connections. As shown in Table 5, only about 30% of the edges in K-Graph are preserved in HNSW.

Besides pruning some "unnecessary" edges, the RNG rule also helps mitigate the skewness of in-degree distribution for high-dimensional vectors. Specifically, the points with high k-occurrence tend to reject more in-links from other points because of the RNG rule. Consider the example in Figure 4 (right), where $C(v) = \{a, b\}$, and the candidate neighbor H has been linked by some points because of closeness. In this case, H will be pruned since it can be reached indirectly from a and b. This example indicates that if a point is close to many other points (i.e., points with high k-occurrence like H), it has to "compete" against these close points for becoming the neighbor of an inserted point. Naturally, the probability of building links with such points is lower than the point with low k-occurrence. This is exactly the opposite to the linking strategy of the K-Graph index. As shown in Table 3, the correlation coefficient between the in-degree and k-occurrence for HNSW is less than 0.3, indicating a very weak correlation. As a result, the inherent skewness of the in-degree distribution of high-dimensional vectors is optimized by the RNG rule: the skewness value drops to nearly a half when compared to K-Graph, as shown in Table 1. Moreover, for every point in HNSW, there exists a sufficient number of short edges for local proximity, but also enough long-range connections for global navigability, which ensure the query efficiency in the recall and navigation stage respectively. As shown in Tables 2 and 4, even though HNSW has a limited out-degree, it uses a similar number of steps to NSW in order to reach a low/high recall. This demonstrates that the RNG rule successfully preserves the navigability in these datasets. As for the overall query performance, HNSW always achieves the best results (well ahead of the second best), across all settings.

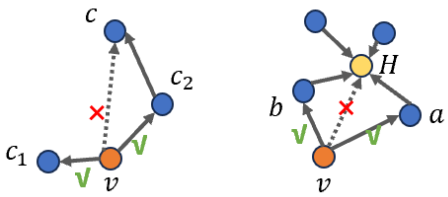
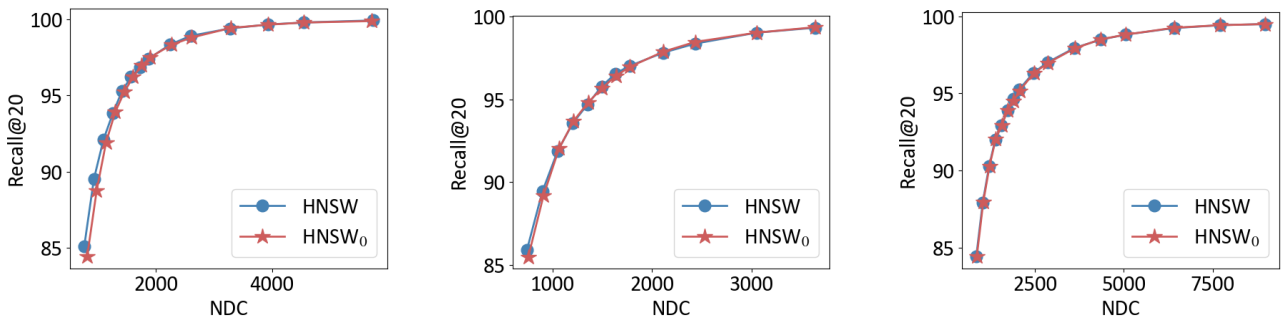


Figure 4: RNG rule.

2.2.4 Hierarchy and Selection of Entry Points

The classical hierarchical index structure of HNSW is widely employed in both research and industrial settings [29, 34, 64, 66]. Intuitively, the upper layers in HNSW form long-range connections providing navigability, while the bottom layer uses small-range links to cover "the last mile". However, we claim that this strategy does not provide any significant benefits in the case of high-dimensional spaces. As shown in Figure 5, using HNSW_0 (the bottom layer of HNSW) alone, leads to nearly the same query performance as (the original, hierarchical) HNSW. This result, which is verified on a dozen of public datasets (omitted due to lack of space) as well as by other studies [43], casts some doubt about the usefulness of the hierarchical structure. We make the following

[3] This is not strict since there is no guarantee on the existence of edge (c_2, c).



(a) Sift1M (b) Deep1M (c) Deep10M

Figure 5: The query performance of HNSW and HNSW$_0$: the difference is negligible.

two observations that try to explain the above result: (i) Due to the distance concentration phenomenon [27], the query is not far away from a random point in the same dataset. For example, in Sift1M and Deep1M datasets, the average distance between a random point and the query is merely **5.3**x and **3.5**x, respectively, larger than the average length of the short edges in K-Graph. (ii) The bottom layer can provide the navigability necessary to quickly approach the local neighborhood. For example, to achieve 99% recall@1, HNSW$_0$ needs 111 hops on the Sift1M dataset, compared to 108 hops for HNSW; on Deep1M, HNSW$_0$ needs 156 hops, compared to 151 hops for HNSW. These small differences verify the efficient navigability of HNSW$_0$.

In this case, the benefit of the selection of entry points for graph-based indexes relies on the accuracy of the entry points and the cost of obtaining these entry points. Unfortunately, the upper layers of HNSW do not show an obvious benefit since they only contain a small portion of data. According to our experiments, only less than half of the queries obtain one of the 50NN in the upper layers, which means that a significant part of the work actually takes place at the bottom layer. To break this limit, subsequent works, including HVS [47] and LSH-APG [81] (both discussed earlier), use another index as an auxiliary structure to help obtain high-quality entry points. The quality of entry points is largely improved, and as reported, HVS achieves over 3x faster query answering time than HNSW on hard datasets and high recall range, while LSH-APG achieves 1.5x faster query answering time, while offering more efficient index construction.

3 Tree-based Index Family

The tree-based index family hierarchically partitions the high-dimensional space, which may be transformed or projected, and groups similar vectors in the same partition as leaf nodes. Usually, the root node of the tree-based indexes covers the whole high-dimensional space and the children nodes of it cover disjoint or overlapped sub-spaces. When querying, we traverse the tree-based index from the root node to one or more leaf nodes by judging which sub-spaces the query vector belongs to or is close to. Then the data inside these leaf nodes are candidates of kNN. In this section, we will survey the latest tree-based indexes proposed in the past five years and summarize the evolution in Figure 6. It can be viewed as an extension of the previous survey of tree-based indexes [51].

3.1 Preliminaries and Overview

The designs of tree-based indexes (in a single machine) can be decomposed into three parts, including data summarization (i.e., dimension reduction), indexing, and querying algorithms. In this subsection, we briefly describe the preliminaries of these techniques leveraged in recent works.

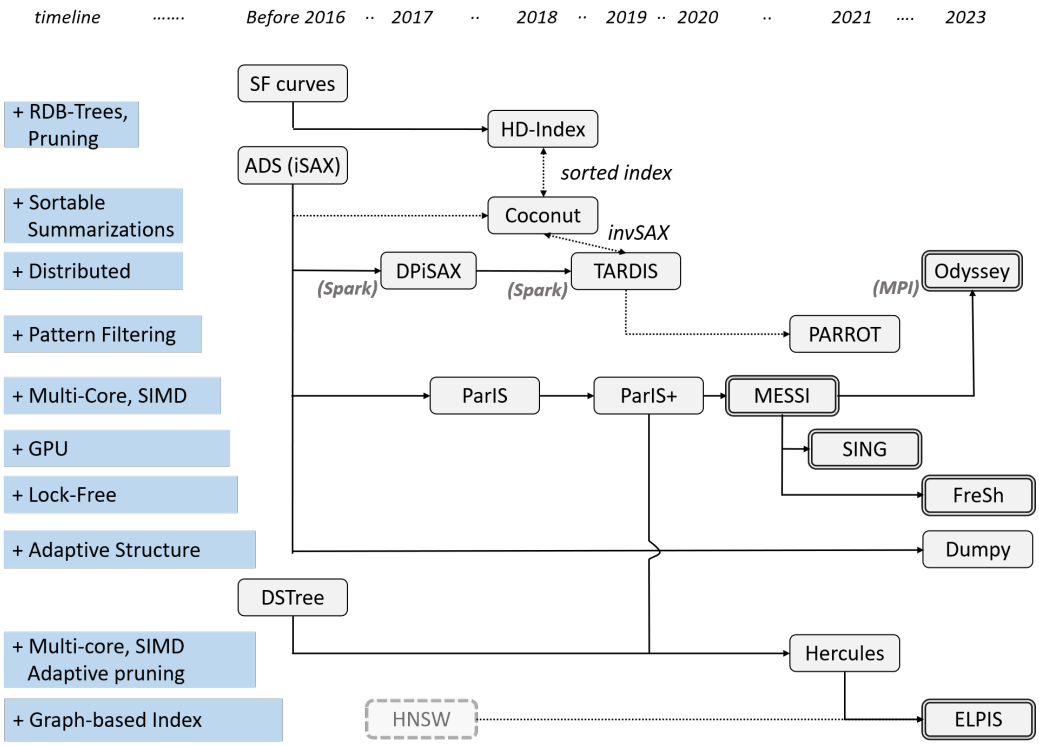


Figure 6: The evolution of tree-based index in the past 6 years, as an extended figure of [51]. The solid arrows denote the inheritance of the index design; the dashed arrows denote the correlation of the design features; indexes in double-lined boxes are in-memory solutions, while the rest are on-disk solutions. Note that HNSW is not a tree-based index; it is included, because ELPIS (discussed in Section 4.2) inherits some of its features.

iSAX-index family iSAX summarization is a dynamic prefix of SAX words [44], and SAX is a symbolization of Piecewise Aggregate Approximation (PAA for short). PAA represents a vector with the means of disjoint equal-length segments, and SAX discretizes PAA with symbols. iSAX word is a prefix of the corresponding SAX word, which can flexibly represent a sub-space in the reduced space. Based on iSAX, the iSAX index family [51] organizes data in a tree structure, where each node is summarized by an iSAX word. The root node covers the whole space and it can be split by refining one segment of its iSAX word. The children nodes own disjoint sub-spaces of their parent. When querying, the iSAX word of each node can be used to prune irrelevant data benefiting from that the distance between iSAX summarization lower bounds the actual distance.

EAPCA-index family The EAPCA-index family [71] differs itself from iSAX by 1) using dynamic segmentation that is determined on the fly when building index, 2) using mean and standard deviation to represent each segment and thus providing tighter bounds when querying, 3) more splitting choices and adaptive split criteria.

Ordered-index family The Ordered-index family [37] sorts high-dimensional data with space-filling curves (e.g., Z-order curve, Hilbert-order curve) and then index them with classical B-tree, etc. Space-filling curves approximately preserve the proximity between points that are close in high-dimensional space. The approximation error increases as the dimensionality goes larger. So usually dimension-reduction techniques are used beforehand.

3.2 Optimized Disk Index Structure

Managing large-scale high-dimensional vectors on disk is challenging since the data locality (i.e., similar vectors should be placed together such that they can be fetched by one I/O) and the index quality (i.e., nearest neighbors should be quickly located by the index) should be considered at the same time. Recent progress has remarkably advanced the state-of-the-art to hour-level indexing time and ms-level query time on billion-sized datasets.

HD-Index [9]. HD-Index, as an ordered index, builds multiple B+ trees each of which stores the Hilbert keys of a segment of dimensions. In this way, HD-Index mitigates the loss of accuracy due to the dimensionality reduction. To facilitate the refinement, HD-Index randomly places pivots and stores in the index the distance between an entry and the nearest reference point, which provides the ability to prune when querying. However, the search cost of HD-Index increases linearly with dimensionality, leading to an inefficient solution for very high-dimensional vectors.

Coconut [37]. In contrast to HD-Index which sorts raw (sub)-vectors, Coconut first reduces dimensionality using SAX, and then sorts SAX words by the z-order curve. In this way, the most significant dimensions (i.e., bits) of the SAX words are considered first. This process matches exactly the principle of SAX summarization, i.e., coarse-grained partitioning is encoded in the most significant bits, while fine-grained partitioning in the least significant bits. Using only one B-tree to index the keys, Coconut has a very fast query time, and the highly packed leaves also lead to a small improvement in the results of approximate search.

Dumpy [67]. Dumpy is an iSAX-based index designed for high-precision approximate search and fast index-building for large datasets. Dumpy leverages the trade-off between the proximity inside the leaf node and the compactness (i.e., the fill factor) of these nodes by dynamically selecting the number of segments and which segments to split along according to the data distribution. It also proposes a leaf node packing algorithm and a vector duplication mechanism that further optimizes the data layout on the index. As a result, Dumpy can provide high-recall answers with much fewer random I/Os.

3.3 Parallel Processing

Parallel indexing and querying on the tree-based index is not trivial for dynamic structure and pruning-based query [54, 57]. The difficulties include the reduction of race conditions and load balancing among threads. The ParIS+ index [57], parallelizes the building and querying algorithms of ADS+ [84], an adaptive variant of the iSAX index. MESSI [54] further optimizes the race conditions for in-memory datasets, while SING [56] extends the query answering algorithm to make use of GPUs. In recent years, the research boundary has been pushed by a parallel and fully-materialized disk index, and a novel lock-free parallelism mechanism.

Hercules [22]. Hercules is an EAPCA-based parallel disk-based index. Designed for large-scale datasets, Hercules optimizes memory management with a two-level buffer model to reduce the time cost of memory management. Besides that, Hercules delicately schedules different tasks with a given number of threads, resulting in (1) CPU-intensive tasks being executed in parallel to I/O-intensive tasks, (2) the number of I/Os being significantly reduced, and (3) avoiding race conditions. Moreover, Hercules proposes a composite query algorithm with hierarchical pruning by different distance approximations. Being the first parallel dynamic (EAPCA-based) tree index, Hercules shows robust and remarkable index-building and query answering performance improvements. (Hercules also inherits properties to ELPIS [23], a parallel in-memory index for approximate search, which we discuss in Section 4.2.)

FreSh [26]. FreSh is an iSAX-based parallel in-memory index, proposing a novel lock-free approach for building and querying the index. FreSh modularizes iSAX-based indexes, and parallelizes the tasks of these modules under Refresh, a generic framework that can be applied on top of any locality-aware data series algorithm to ensure lock-freedom. FreSh also designs a helping mechanism with a small synchronization cost for load balancing among threads. The combination of the tree-based index and lock-free mechanism opens more directions for improving the parallelism of tree-based indexes.

3.4 Distributed Processing

The core challenges of distributed indexes are how to place the dataset into different machines, how to achieve load balancing among these machines, and how to coordinate these machines to serve queries. The previous study in this area, i.e., DPiSAX [74], partitions data by the distribution on the SAX space of the sampling data and dispatches queries to corresponding partitions. Recent studies have advanced it from the aspects of partition quality, load balance, and scalability of indexing and querying, by introducing and designing novel techniques for distributed systems.

TARDIS [79]. As an iSAX-based index, the rationale of TARDIS is to cluster similar nodes into the same pack. The pack is organized as middle-sized and the placement of these packs depends on the downstream distributed file systems (e.g., HDFS). The clustering is based on a variant of iSAX, invSAX, which places the prior bits into the first positions and then the later bits. TARDIS prominently advances DPiSAX [74] by load balancing and cooperative querying. However, it loses the ability to prune and suffers from the low proximity of data inside a pack, limiting the practicability and the improvement.

PARROT [76]. PARROT is built as a secondary index for a partitioned data warehouse. PARROT identifies patterns by clustering on dimension-reduced SAX space, and stores the distribution of patterns in a global index with exception points. However, PARROT relies on the existence of a strong correlation pattern between the partition key and the vector inside. Otherwise, the vectors of the same pattern will distribute sparsely across the data warehouse and querying will deteriorate to a linear scan.

Odyssey [13]. Odyssey is a scalable framework for distributed in-memory data series similarity search in clusters with multi-core servers, thus, exploiting parallelization both inside and across system nodes. Odyssey follows the classical design principle of horizontal splitting and vertical duplication in distributed systems. Unlike TARDIS, Odyssey tries to achieve load balancing through carefully partitioning the data across machines (by storing similar series to different machines), assigning queries to machines using a clever scheduling algorithm (by estimating the execution time of each query), and employing more resources for hard queries (by using a light-weight work-stealing technique). Odyssey also duplicates the data splits across several machines, which enables further parallelization in query answering, as well as load-balancing (through work-stealing). As a result, Odyssey achieves nearly linear scalability on an increasing number of machines, or queries.

3.5 Discussion

From the summarization and analyses above, we can observe that tree-based indexes are developed toward practical, large-scale, parallel, and distributed scenarios. These extensions fully leverage the data locality and high efficiency of tree-based indexes. On the other hand, an inherent drawback of the tree-based indexes is that it is hard to overcome the curse of dimensionality. The similarity between the points in a partition of a tree index is decided by a hyperplane in high dimensions, which usually needs to balance separability and the proximity of the data. As the height of a tree is limited (which influences the search performance), only limited dimensions and

Table 6: A comparative study of graph- and tree-based indexes.

Comparison Items		Graph	Tree
Basic Properties	Rationale	Navigability	Space-partitioning
	Data locality	×	✓
	Dimension reduction	×	✓
Query Answering	Search pattern	Best-first-search	Prioritized tree traversal
	Two-stage search	✓	✓
	Navigating stage	Fast	Fast
	Recall stage	Fast	Slow
	Progressive search	✓	✓
	Pruning/ ϵ-accuracy guarantee	×	✓
	Early termination of distance calculation	✓	✓
Parallelism	Index construction	✓	✓
	Intra-query parallelism	✓	✓
Distributed indexing	Load balance	×	✓
	Query collaboration	×	✓

features are considered to form the hyperplane. Therefore, it is not easy to quickly locate all the nearest neighbors in a tree index.

4 Comparative Study and Open Directions

4.1 Comparison Between Graph- and Tree-based Indexes

As shown in Table 6, we compare graph- and tree-based indexes along four dimensions, namely, basic properties, query answering, parallelism, and distributed indexing.

Basic properties. The motivations of these two kinds of indexes are different. Graph-based indexes are motivated by the navigability of a small-world graph model whereas tree-based indexes organize data according to the proximity of neighboring vectors. Therefore, tree-based indexes group similar vectors together and own data locality while graph-based indexes are not aware of the data locality explicitly. Since graph-based indexes only use the absolute distance between points for indexing, the dimensions of the vectors are usually ignored and hence no dimension reduction techniques are used, despite that the dimensions of space are impacting the quality of indexes implicitly. On the contrary, tree-based indexes need to explicitly split the partitions and a high dimensionality will severely impact the quality of splits. Thus, dimensionality reduction has become the rule of thumb for tree-based indexes.

Tree-based indexes can usually be deployed either on disk or in memory, and enjoy cache acceleration by virtue of the data locality. However, it is hard for graph-based indexes to leverage data locality as they are secondary indexes (i.e., the data is not stored inside the index). Moreover, inserting a vector into a graph-based index usually means a change in the topology of the graph. This indicates that ingestion throughput is limited, especially when the graph is large. Some promising studies propose efficient on-disk solutions [15, 64, 80]

and effective update strategies [63] to address these problems, which nevertheless, remain interesting research directions for graph-based indexes.

Query answering. As for querying, graph-based indexes follow a best-first-search pattern on the graph, while tree-based indexes find the closest partitions to the query vector by prioritized tree traversal. Nevertheless, both these two algorithms are reminiscent of two-stage searching [77] (i.e., navigating and recall stages). For both algorithms, the navigating stage can be very fast while in the recall stage, graph-based indexes are far more efficient than the tree-based ones. This is because the kNN of the candidates are directly linked in the graph index. On the other hand, the tree-based index provides a bound for the query vector and a group of database vectors, allowing multi-level pruning with no false negatives, which can also be used to build mechanisms that provide (deterministic or probabilistic) guarantees for the approximate search answers [24, 32]. On the contrary, graph-based indexes cannot safely prune any of the visited points. As a common point, when pruning fails, both graph- and tree-based indexes can early stop the exact distance calculations between the query and the candidate answers (i.e., without having to process all the dimensions) [31, 60]. Note that while graph-based indexes can only support ANNS (with no quality guarantees), the tree-based indexes discussed in this paper support all flavors of similarity search, ranging from ANNS without and with (probabilistic and deterministic) quality guarantees to exact similarity search.

Another direction that has been studied in the context of both graph- and tree-based indexes is progressive search [32, 35, 42, 78]. In the recall stage, ANNS tries to answer two questions: (i) Is the current Best-So-Far (BSF) answer the exact kNN? (ii) If not, when will the exact kNN occur? Note that the answers to these questions are not known during query answering. However, estimations of the answers to these questions can benefit the query algorithms. First, if the answer to the first question is positive, the query algorithm can terminate immediately. As demonstrated in an earlier study [32], the time needed for ANNS to determine the answer to the first question after having already found the exact kNN, corresponds to the vast majority of the total query answering time. Therefore, an agile termination can significantly accelerate high-precision search. Recently, ProS [32] proposed the use of simple machine learning models to provide various probabilistic quality guarantees (such as, on the current result being the exact kNN, the distance of the current answer being less than ϵ from the exact kNN distance, and others) for intermediate (i.e., progressive) results during the query answering process. Moreover, by answering the second question, we can dynamically schedule the computing resources by controlling the search parameters, such as ef, and the number of leaves to visit for a precision target. To achieve this, for tree-based indexes, ProS uses quantile regression that estimates the $1-\phi$ quantile of the time needed to find the exact answer with the BSF answers. For graph-based indexes, a recent study [42] enriches the features by (a) the query itself, (b) the progress made from the start point to the current point, and (c) the ratio of existing features. With these features, it trains Gradient Boosting Decision Trees for regression.

Parallelism. Both kinds of indexes have been extended to support parallelization. For index construction, tree-based indexes have evolved to the solutions described above, and such parallelism for graph-based indexes is also natural. For intra-query parallelism, these two kinds of index families also propose corresponding extensions, like Hercules [22], MESSI [54], SING [56], FreSh [26] and ELPIS [23] for tree-based indexes, and SONG [82] and iQAN [58] for graph-based indexes.

Distributed indexing. While some studies have focused on distributed solutions for tree-based indexes [13, 74, 76, 79], the direction of distributed graph-based indexes has not been studied in depth. Existing solutions [29, 66] use random data partitioning and independent building of the corresponding parts of the index. Therefore, the computing power of the machines is not fully leveraged, and there are untapped opportunities for improving scalability [21, 78].

4.2 Existing Combinations of the Two Kinds of Indexes

In this subsection, we survey the existing solutions that combine these two kinds of indexes and analyze their common points and design principles.

SPTAG [16] and SPANN [15]. Similar to HVS and LSH-APG, SPTAG adopts a balanced K-means tree [45] to locate better entry points. While SPANN as a disk index, partitions data by a balanced K-means tree, and each partition is stored sequentially on disk. The data partitions are further refined by duplication. To quickly find the nearest partitions, SPANN builds a SPTAG index in memory for the partition centroids. As a result, it shows remarkable improvement compared with dumping an optimized graph into the disk [64].

LANNS [20] and ELPIS [23]. Another family of solutions constructs a tree index to partition the data, and builds a graph within each partition (leaf node). LANNS uses a learned hyperplane for splitting and a multi-path search in the tree. ELPIS [23] leverages Hercules [22] to partition the data. In contrast to LANNS, ELPIS can prioritize the leaf node according to the lower bound distance to the EAPCA of the query. In this way, ELPIS provides higher intra-query parallelism, leading to a remarkable improvement in query performance on large-scale datasets. Since the indexing cost of the tree index is lower than a graph index, ELPIS significantly improves indexing efficiency by 3x-8x, and reduces memory consumption by 40%. Interestingly, ELPIS shows the trade-off between query efficiency and accuracy when varying the number of partitions. That is, more partitions improve search efficiency at the expense of accuracy, but there exists a sweet point that optimizes performance.

Discussion. So far we have studied three kinds of combinations of graph- and tree-based indexes, including (1) using the tree index to obtain better entry points for the graph index (HVS, LSH-APG, and SPTAG); (2) using the graph index to find the closest leaf nodes in the tree index (SPANN); (3) using the tree index to partition data and building graph indexes on each leaf node (LANNS and ELPIS). We can deduce the following key ideas from these combinations. (i) The tree index can promote the representation granularity from vector-level to node level, even in sub-tree level, which makes it easy to handle large-scale and disk-resident datasets. (ii) The cases in large-sized datasets (billion-level) are not the same as middle-sized (million-level), where the pros and cons of different index families should be re-considered carefully. (iii) The tree index still suffers from the boundary issue and must be adapted to improve the search accuracy (e.g., duplication). (iv) It is yet intractable to build the graph index on the whole dataset, while at the same time, the graph index is irreplaceable for achieving high recall efficiently. Based on these explorations, in the next section, we will introduce more possible directions where tree-based, as well as other kinds of indexes can assist the graph index to overcome more inherent limitations.

4.3 Open Problems and Promising Future Directions

In this subsection, we propose four open problems that are crucial for solving ANNS problems in practice and at scale. More importantly, combining the techniques from the tree- and graph-based index is a promising way to solve these problems.

Dimension reduction and pruning. The most costly operation for graph-based indexes is the distance calculation, reducing dimensions can provide a linear improvement on query [31]. For example, by rotating the vectors in the directions of the principal component vectors, the early termination mechanisms can behave more efficiently as most of the variations are concentrated in the first dimensions. The results are shown in Figure 7, where the query time is reduced to **50**% with this simple optimization. Moreover, representation learning is also a promising way of embedding high-dimensional vectors in a low-dimensional space. A lower bound of the distance measure in the embedded space is often designed at the same time, in order to safely perform pruning. As demonstrated by GRAIL [53] and SEAnet [70], representation learning is more flexible in capturing the "shape" of

high-dimensional vectors than traditional methods, because it can adapt to the data at hand. Similar data-adaptive techniques have been studied in the context of quantization [7, 52]. Besides that, inspired by the hierarchical structure of tree-based indexes, after dimension reduction (or rotation, projection), it would be helpful if different vectors can be summarized with a common representation and be safely pruned, i.e., through node-level pruning.

Distributed indexing. Distributed indexing is a must for managing large-scale high-dimensional data in production. Distributed tree-based indexes, like Odyssey, have been designed for exact kNN search; their ANNS performance can be further improved. On the other hand, partitioning a graph index is inherently difficult since it is not evident how to partition the graph with load-balanced partitions. As demonstrated in [78], combining the results from multiple disjoint graph indexes split at random is sub-optimal for this problem, and other clustering algorithms like k-means, suffer from imbalance problems. Therefore, designing a load-balance-oriented data partition scheme that minimizes the loss of efficiency will provide substantial improvements to vector database performance [3, 66, 72]. In this case, a promising direction is to leverage the space-partitioning scheme of tree-based indexes for splitting data, while estimating the load of each split dynamically, in order to provide more opportunities for collaborative (parallel) work during query answering time.

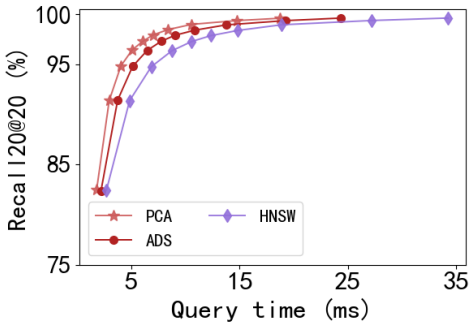


Figure 7: Early termination techniques in HNSW on Gist1M. ADS [31] is the state-of-the-art technique.

Reliable ANNS. Current ANNS algorithms are not reliable. In different datasets (similar cardinality and dimensionality), the search performance is significantly different. For example, to achieve 90% recall@20 on HNSW, the search cost is 90x larger for Glove dataset than Deep1M dataset. Such a result is not a special case [31, 43, 55, 68]. The reason is closely related to our analyses in Section 2. Although HNSW removes the hubs and mitigates the skewness, the effect of these techniques actually depends on the distribution of the original dataset. If the distribution of the original dataset is very skewed, the phenomenon that hubs are clustered to trap the search route occurs again in HNSW. It means that the high efficiency for navigation, and the high local proximity for recalling no longer exist in graph-based indexes. To further mitigate the skewness, it will be helpful to use (approximate) range search to further filter edges. It is possible to remove the bias on the nodes with high k-occurrence and provide a robust performance over datasets of any distribution.

Reliable benchmarks. With the rapid growth of the ANNS area, the development of appropriate benchmarks is urgently required, in order to evaluate the solutions based on the different index families we outlined earlier, as well as new solutions that combine and extend the ideas of existing techniques. Some works in the literature [24, 25, 41] have been pioneering in this respect, but the latest techniques are not included in them. Other benchmarks such as [62] and [10, 11] provide an open framework to evaluate the performance of different ANNS algorithms. However, many engineering tricks can be added to improve the literal performance which on the other hand, might confuse readers without detailed ablation studies. Moreover, common public datasets such as Sift [6], Gist [6], Deep [5], were produced over a decade ago representing the application of ANNS in the machine learning era. As the model architecture has evolved significantly in the past decade, it would be beneficial to produce and use newer datasets that correspond to modern applications (e.g., Transformer-based models).

Furthermore, it would be interesting to assess the difficulty of similarity search queries: the different difficulty levels represented by various datasets, as well as various queries on the same dataset, have been observed in the

literature [10, 85]. However, we still lack a comprehensive analysis and theoretical characterization of the difficulty level of datasets and queries alike, which directly affect the performance of the various indexing approaches, and may also lead to biased evaluation results. Previous work has studied local intrinsic dimensionality [10], as well as measures related to the tightness of lower bound and data distribution [85], in order to evaluate the difficulty of a query and a dataset, or more precisely, of a query with respect to a given dataset. These approaches study the difficulty of pruning candidate points in the dataset, in response to a given kNN query. Nevertheless, more extensive and comprehensible studies and benchmarks are needed to more accurately and fairly characterize and evaluate the performance of different ANNS indexes.

5 Conclusions

In this paper, we study the evolution of the graph-based index family with ablation studies and observe that the keys to its success lie in the randomization and the RNG-based pruning rule. These techniques preserve the local proximity with fewer edges while building effective long-range connections to enhance navigability. We also survey the recent progress of the tree-based index family and discuss their relationships. Moreover, we conduct a comparative study over these two index families and observe that several of these techniques can complement each other in solving practical problems in ANNS. Finally, we point to some interesting research directions in the context of ANNS.

References

[1] OpenAI. Chatgpt-retrieval-plugin. https://github.com/openai/chatgpt-retrieval-plugin, Accessed Sep. 3, 2023.

[2] Pgvector. https://github.com/pgvector/pgvector/ Accessed Sep. 4, 2023.

[3] Pinecone. https://www.pinecone.io/ Accessed Sep. 4, 2023.

[4] Scikit-learn: Machine Learning in Python. https://scikit-learn.org/ Accessed Sep. 4, 2023.

[5] Skoltech Computer Vision. Deep Billion-Scale Indexing. http://sites.skoltech.ru/compvision/noimi, 2018.

[6] TEXMEX Research Team. Datasets for Approximate Nearest Neighbor Search. http://corpus-texmex.irisa.fr/, 2018.

[7] C. Aguerrebere, I. Bhati, M. Hildebrand, M. Tepper and T. Willke. Similarity Search In The Blink Of An Eye With Compressed Indices. PVLDB, 2023.

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

[9] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya. Hd-index: Pushing The Scalability-accuracy Boundary For Approximate KNN Search In High-dimensional Spaces. PVLDB, 2018.

[10] M. Aumüller and M. Ceccarello. The Role of Local Dimensionality Measures in Benchmarking Nearest Neighbor Search. Inform. Syst., 2021.

[11] M. Aumüller, E. Bernhardsson and A. Faithfull. ANN-Benchmarks: A Benchmarking Tool For Approximate Nearest Neighbor Algorithms. Inform. Syst., 2020.

[12] L. Cayton. Fast Nearest Neighbor Retrieval for Bregman Divergences. ICML, 2008.

[13] M. Chatzakis, P. Fatourou, E. Kosmas, T. Palpanas, and B. Peng. Odyssey: A Journey In The Land of Distributed Data Series Similarity Search. PVLDB, 2023.

[14] D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading Wikipedia to Answer Open-domain Questions. ACL, 2017.

[15] Q. Chen, B. Zhao, H. Wang, M. Li, C. Liu, Z. Li, M. Yang and J.Wang SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search. NeurIPS, 2021.

[16] Q. Chen, H. Wang, M. Li, G. Ren, S. Li, J. Zhu, J. Li, C. Liu, L. Zhang, and J. Wang. SPTAG: A Library for Fast Approximate Nearest Neighbor Search. https://github.com/Microsoft/SPTAG, 2018.

[17] P. Chen, W. Chang, J. Jiang, H. Yu, I. Dhillon, and C. Hsieh FINGER: Fast Inference for Graph-based Approximate Nearest Neighbor Search. WWW, 2023.

[18] A. Das, M. Datar, A. Garg, and S. Rajaram. Google News Personalization: Scalable Online Collaborative Filtering. WWW, 2007.

[19] W. Dong, C. Moses, and K. Li. Efficient K-nearest Neighbor Graph Construction for Generic Similarity Measures WWW, 2011.

[20] I. Doshi, D. Das, A. Bhutani, R. Kumar, R. Bhatt, and N. Balasubramanian. LANNS: A Web-scale Approximate Nearest Neighbor Lookup System. PVLDB, 2022.

[21] S. Deng, X. Yan, K. Kelvin, C. Jiang, and J. Cheng. Pyramid: A General Framework for Distributed Similarity Search on Large-scale Datasets. BIGDATA, 2019.

[22] K. Echihabi, P. Fatourou, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Hercules Against Data Series Similarity Search. PVLDB, 2022.

[23] K. Echihabi, K. Zoumpatianos, and T. Palpanas. ELPIS: Graph-Based Similarity Search for Scalable Data Science. PVLDB, 2023.

[24] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Return of The Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. PVLDB, 2019.

[25] 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. PVLDB, 2018.

[26] P. Fatourou, E. Kosmas, T. Palpanas and G. Paterakis. FreSh: A Lock-Free Data Series Index. SRDS, 2023.

[27] D. Francois, V. Wertz, and M. Verleysen. The Concentration of Fractional Distances. TKDE, 2007.

[28] C. Fu and D. Cai. EFANNA: An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph. arXiv, 2016.

[29] C. Fu, C. Xiang, C. Wang, and D. Cai. Fast Approximate Nearest Neighbor Search with The Navigating Spreading-out Graph. PVLDB, 2019.

[30] C. Fu, C. Wang, and D. Cai. High Dimensional Similarity Search with Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. TPAMI, 2021.

[31] J. Gao, and C. Long. High-dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations. SIGMOD, 2023.

[32] A. Gogolou, T. Tsandilas, K. Echihabi, A. Bezerianos, and T. Palpanas. ProS: Data Series Progressive k-NN Similarity Search and Classification with Probabilistic Quality Guarantees. VLDBJ, 2023.

[33] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang. Retrieval Augmented Language Model Pre-training. PMLR, 2020.

[34] N. Hezel, K. Barthel, K. Schall, and K. Jung. Fast Approximate Nearest Neighbor Search with a Dynamic Exploration Graph Using Continuous Refinement. arXiv, 2023.

[35] J. Jo, J. Seo, and J. Fekete. PANENE: A Progressive Algorithm for Indexing and Querying Approximate K-Nearest Neighbors. TVCG, 2018.

[36] J. Johnson, M. Douze, and H. Jégou. Billion-scale Similarity Search with GPUs. TBD, 2019.

[37] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut: A Scalable Bottom-up Approach for Building Data Series Indexes. PVLDB, 2018.

[38] M. Li, Y. Zhang, Y. Sun, W. Wang, I. W. Tsang and X. Lin. I/O Efficient Approximate Nearest Neighbour Search Based on Learned Functions. ICDE, 2020.

[39] K. Li and J. Malik. Fast k-nearest Neighbour Search via Prioritized DCI. ICML, 2017.

[40] H. Li, Y. Su, D. Cai, Y. Wang, and L. Liu. A Survey on Retrieval-Augmented Text Generation. arXiv, 2022.

[41] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, And Improvement. TKDE, 2019.

[42] C. Li, M. Zhang, D. Andersen, and Y. He. Improving Approximate Nearest Neighbor Search Through Learned Adaptive Early Termination. SIGMOD, 2020.

[43] P. Lin and W. Zhao. Graph-based Nearest Neighbor Search: Promises and Failures. arXiv, 2019.

[44] J. Lin and E. Keogh. A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. DMKD, 2003.

[45] H. Liu, Z. Huang, Q. Chen, M. Li, Y. Fu, and L. Zhang. Fast Clustering with Flexible Balance Constraints. BIGDATA, 2018.

[46] J. Liu, Z. Zhu, J. Hu, H. Sun, L. Liu, L. Liu, G. Dai, H. Yang, and Y. Wang. Optimizing Graph-based Approximate Nearest Neighbor Search: Stronger and Smarter. MDM, 2022.

[47] K. Lu, M. Kudo, C. Xiao, and Y. Ishikawa. HVS: Hierarchical Graph Structure Based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. PVLDB, 2022.

[48] Y. Malkov and D. Yashunin. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. TPAMI, 2018.

[49] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate Nearest Neighbor Algorithm Based on Navigable Small World Graphs Inform Syst, 2014.

[50] Y. Matsui, Y. Uchida, H. jégou, and S. Satoh. A Survey of Product Quantization. ITE Transactions on Media Technology and Applications, 2018.

[51] T. Palpanas. Evolution of A Data Series Index: the iSAX Family of Data Series Indexes. CCIS, 2020.

[52] J. Paparrizos, I. Edian, C. Liu, A. J. Elmore and M. Franklin. Fast Adaptive Similarity Search Through Variance-Aware Quantization. ICDE, 2022.

[53] J. Paparrizos and M. Franklin. GRAIL: Efficient Time-series Representation Learning. PVLDB, 2019.

[54] B. Peng, P. Fatourou, and T. Palpanas. MESSI: In-memory Data Series Indexing. ICDE, 2020.

[55] Y. Peng, B. Choi, T. Chan, J. Yang, and J. Xu. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. SIGMOD, 2023.

[56] B. Peng, P. Fatourou, and T. Palpanas. SING: Sequence Indexing Using GPUs. ICDE, 2021.

[57] B. Peng, P. Fatourou, and T. Palpanas. ParIS+: Data Series Indexing on Multi-core Architectures. TKDE, 2020.

[58] Z. Peng, M. Zhang, K. Li, R. Jin and B. Ren iQAN: Fast and Accurate Vector Search with Efficient Intra-Query Parallelism On Multi-Core Architectures. PPoPP, 2023.

[59] M. Radovanovic, A. Nanopoulos and M. Ivanovic. Nearest Neighbors in High-dimensional Data: the Emergence and Influence of Hubs. ICML, 2009.

[60] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and Mining Trillions of Time Series Subsequences Under Dynamic Time Warping. KDD, 2012.

[61] C. Silpa-Anan and R. Hartley. Optimised KD-trees for Fast Image Descriptor Matching. CVPR, 2008.

[62] H. Simhadri, G. Williams, M. Aumüller, M. Douze, A. Babenko, D. Baranchuk, Q. Chen, L. Hosseini, R. Krishnaswamy, G. Srinivasa, S. Subramanya and J. Wang. Results of the NeurIPS'21 Challenge on Billion-scale Approximate Nearest Neighbor Search. NeurIPS (Competition and Demos), 2021.

[63] A. Singh, S. Subramanya, R. Krishnaswamy, and H. Simhadri FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search. arXiv, 2021.

[64] S. Subramanya, Devvrit, R. Kadekodi, R. Krishaswamy and H. Simhadri. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on A Single Node. NeurIPS, 2021.

[65] G. Toussaint. The Relative Neighbourhood Graph of A Finite Planar Set. Pattern recognition, 1980.

[66] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, K. Yu, Y. Yuan, Y. Zou, J. Long, Y. Cai, Z. Li, Z. Zhang, Y. Mo, J. Gu, R. Jiang, Y. Wei, C. Xie. Milvus: A Purpose-built Vector Data Management System. SIGMOD, 2021.

[67] Z. Wang, Q. Wang, P. Wang, T. Palpanas, and W. Wang. Dumpy: A Compact and Adaptive Index for Large Data Series Collections. SIGMOD, 2023.

[68] H. Wang, Z. Wang, W. Wang, Y. Xiao, Z. Zhao, and K. Yang. A Note on Graph-based Nearest Neighbor Search. arXiv, 2020.

[69] M. Wang, X. Xu, Q. Yue, and Y. Wang. A Comprehensive Survey and Experimental Comparison of Graph-based Approximate Nearest Neighbor Search. PVLDB, 2021.

[70] Q. Wang and T. Palpanas. SEAnet: A Deep Learning Architecture for Data Series Similarity Search. TKDE, 2023.

[71] Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A Data-adaptive and Dynamic Segmentation Index for Whole Matching on Time Series. PVLDB, 2013.

[72] C. Wei, B. Wu, S. Wang, R. Lou, and C. Zhan. AnalyticDB-V: A Hybrid Analytical Engine towards Query Fusion for Structured and Unstructured data. PVLDB, 2020.

[73] X. Xu, M. Wang, Y. Wang, and D. Ma. Two-stage Routing with Optimized Guided Search and Greedy Algorithm on Proximity Graph. KBS, 2021.

[74] D. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. Massively Distributed Time Series Indexing and Querying. TKDE, 2018.

[75] Y. Yu, D. Wen, Y. Zhang, L. Qin, W. Zhang, and X. Lin. GPU-accelerated Proximity Graph Approximate Nearest Neighbor Search and Construction. ICDE, 2022.

[76] L. Zhang, N. Alghamdi, H. Zhang, M. Eltabakh, and E. Rundensteiner. PARROT: Pattern-based Correlation Exploitation In Big Partitioned Data Series. VLDBJ, 2022.

[77] 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.

[78] P. Zhang, B. Yao, C. Gao, B. Wu, X. He, F. Li, Y. Lu, C. Zhan, and F. Tang. Learning-based Query Optimization for Multi-probe Approximate Nearest Neighbor Search. VLDBJ, 2012.

[79] L. Zhang, N. Alghamdi, M. Eltabakh, and E. Rundensteiner. TARDIS: Distributed Indexing Framework for Big Time Series Data. ICDE, 2019.

[80] M. Zhang and Y. He. GRIP: Multi-Store Capacity-Optimized High-Performance Nearest Neighbor Search for Vector Search Engine. CIKM, 2019.

[81] X. Zhao, Y. Tian, K. Huang, B. Zheng, and X. Zhou. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-dimensional Spaces. PVLDB, 2023.

[82] W. Zhao, S. Tan, and P. Li. SONG: Approximate Nearest Neighbor Search on GPU. ICDE, 2020.

[83] B. Zheng, X. Zhao, L. Weng, Q. Nguyen, H. Liu, and C. Jensen. PM-LSH: A Fast and Accurate In-memory Framework for High-dimensional Approximate NN and Closest Pair Search. VLDBJ, 2021.

[84] K. Zoumpatianos, S. Idreos, and T. Palpanas. ADS: The Adaptive Data Series Index. VLDBJ, 2016.

[85] K. Zoumpatianos, Y. Lou, I. Ileana, T. Palpanas, and J. Gehrke. Generating Data Series Query Workloads. VLDBJ, 2018.

Exploiting Modern Hardware Architectures for High-Dimensional Vector Search at Speed and Scale

Minjia Zhang*, Jie Ren‡*, Zhen Peng†, Ruoming Jin‡, Dong Li†*, Bin Ren‡*
*Microsoft, minjiaz@microsoft.com
†Pacific Northwest National Laboratory
‡Kent State University
†*University of California, Merced
‡*College of William & Mary

Abstract

The field of vector search has seen a surge in interest from both researchers and practitioners due to its potential in emerging AI applications. Understanding how to optimize its performance is crucial for numerous tasks, but there remain a lot of challenges in practice. The advent of new hardware architectures and platforms has prompted a reevaluation of the design of large-scale vector search systems. However, current state-of-the-art vector search algorithms have not fully leveraged new hardware architectures to maximize performance.

In this study, we propose design strategies to enhance the computational and memory efficiency of large-scale vector search. Our novel search algorithm, iQAN, delivers up to an order of magnitude faster search speeds on multi-core architectures through efficient intra-query parallelism, effectively utilizing the combined computational power of modern multi-core chips. Our new design, HM-ANN, employs a novel form of index that effectively leverages heterogeneous memory, enabling billion-scale vector search at a low cost. This paper delves into the challenges and algorithms associated with iQAN and HM-ANN, with a focus on improvements in computational and memory efficiency. The paper also includes the results of our experiments that demonstrate the outstanding performance of vector search when modern hardware architectures are effectively utilized through our proposed methods. Lastly, the paper explores open questions and future directions for supporting high-dimensional vector search with speed and scale.

1 Introduction

1.1 Approximate Nearest Neighbor Search

Finding the top-k nearest neighbors among database vectors for a query has long been a key building block to solve problems such as large-scale information retrieval and image search [33, 38, 49], recommendation [17], entity resolution [27], and sequence matching [10]. As database size and vector dimensionality increase, exact nearest neighbor search becomes expensive and impractical due to latency and memory constraints [11, 12, 62]. Therefore, to reduce the search cost, various approximate nearest neighbor search (ANNS) algorithms have been proposed to improve efficiency substantially while mildly relaxing accuracy constraints, leading to the so-called accuracy-vs-efficiency tradeoffs.

Over the years, a variety of algorithms for Approximate Nearest Neighbor Search (ANNS) have been developed with the goal of enhancing computational and memory efficiency. To improve the compute efficiency, well-designed indexes have been introduced, including tree structure-based [8, 9, 44], hashing-based [26], and proximity graph-based approaches [24, 40]. To improve memory efficiency, various compression algorithms have also been applied to ANNS, such as product quantization-based methods [25, 31, 32, 36, 46]. These methods

can also be combined to improve both compute and memory efficiency simultaneously. For a more detailed understanding and comparison of ANNS algorithms, we recommend several literature that provide excellent surveys [3, 37, 58].

1.2 Modern Applications and Requirements

ANNS holds significant relevance in contemporary applications, particularly in conjunction with deep learning models, as it facilitates innovative search scenarios. Traditional entity retrieval relies on keyword matching and user behavior signals. However, with the progression of deep learning, it is now possible to construct models that yield vectors with close distances for entity inputs sharing similar "views". With these models, one then can encode unstructured data into embedding vectors in a high dimensional space $\mathbb{R}^d$ [28, 65]. These vectors capture the similarities between various entities within the latent space. As a result, the nearest embeddings for a specific query often symbolize entities with similar semantics in the latent space. ANNS then emerges as a natural choice for managing these vectors while ensuring both speed and accuracy in retrieval.

Vector-based search has already been integrated into many modern applications. For instance, Web-scale search engines like Google [56] and Bing [14, 55] utilize embeddings for documents (e.g., word2vec [43] and doc2vec [34]) and images (e.g., VGG [54]) to retrieve semantically related entities in response to user queries. Major e-commerce players like Amazon [45] have developed recommendation systems that embed both the product catalog and the search query, recommending products whose embeddings are closest to the embedded search query. YouTube has built search engine that embeds videos to vectors for video recommendation [16]. More recently, vector search has been employed in retrieval augmented generation in large language models (LLMs), where vector search can be used to expand LLMs knowledge by incorporating external data sources [13]. Vector search also presents a fertile ground for exploring future applications. For instance, recent advancements in deep learning have enabled models to capture multimodal relationships, such as through the use of multimodal foundation models [21]. Consequently, the underlying vector search systems can also leverage ANNS to handle multi-modality entities. However, how to effectively handle different modalities and capture the full range of interconnections and relationships among them via ANNS remains an open question. This includes whether various modalities benefit from using the same or different vector search methods, which is an exciting area for future exploration.

As vector search goes to a larger scale, where the dimension scales from $\sim$100 to $\sim$1000 and the number of vectors scales from millions to billions, the challenge of serving latency becomes more prominent even with novel ANNS algorithms. For instance, online interactive services (e.g., web search engine) often require responses to be returned within a few or tens of milliseconds, as delayed responses could degrade user satisfaction and affect revenue [22]. However, as the number of entities (such as images and documents) grows rapidly and deep learning embeddings expand to higher dimensions (from embedding sentences to full documents), it becomes increasingly difficult to find highly accurate results in large datasets while adhering to latency constraints. Many vector search services, such as text and image search, require intensive computation and may not be feasible due to latency violations. Therefore, how to transform these applications from impossible to ship due to latency violation to well-fitting SLA is crucial for the practical adoption of vector search. Another big requirement for large-scale vector search is cost reduction. Large-scale services deal with a vast volume of requests and could necessitate thousands of machines for a single application. Therefore, decreasing the number of machines while maintaining the same search quality and latency is crucial for reducing the total cost of ownership for the application.

1.3 New Hardware Architectures and Opportunities

Existing ANN algorithms have mostly exploited the uni-core CPU infrastructure and standard memory hierarchy. This infrastructure uses processors whose performance increased with Moore's Law, thus limiting the need for high levels of concurrent execution on a single machine. However, processors are no longer providing ever

higher uni-core performance. Meanwhile, the prior infrastructure used DRAM for main memory. However, the main memory capacity is often quite limited to hold a large volume of data. As multi-core processors become ubiquitous and new memory architecture such as heterogeneous memory becomes available, new opportunities for large-scale vector search exist:

- **Design for multi-core**: Modern CPUs are often equipped with high-performance multi-core. Since uni-core speed has pretty much saturated, we need to get better at exploiting a large number of cores by addressing at least two important aspects:

  1. Multi-core CPUs provide high concurrency, but as the level of concurrency increases, synchronization among different cores are more likely to block and limit scalability.
  2. The performance of multi-core processes also depend on the shared memory bandwidth utilization. According to the roofline model [64], the performance of an application is not only bounded by the compute capability but also the bandwidth performance. So how to make the best utilization of memory bandwidth needs great care.

- **Design for modern memory devices**: Vector search at large scale is very memory consuming and easily runs out of memory with a few hundred millions of vectors. When the dataset becomes too large to fit on a single machine, one approach is to use the compressed representations of the database points, such as Hamming codes [47] and product quantization [19, 25, 31, 32, 46]. However, the performance of these methods deteriorates rapidly at higher recall targets, because they calculate approximate distance based on compressed vectors instead of on the original data vectors. Another approach is to exploit storage. In DiskANN [55], the authors explore slow storage to achieve billion-scale ANNS in a single machine. However, disk latency is a major problem. While persistent media such as SSD offers lower latency and much higher I/O ops per second than traditional disks, they are still several orders of magnitude slower than DRAM. Based on this assumption, data access to the persistent media during search should be minimized. As a result, DiskANN maintains a copy of compressed data in memory with product quantization [55], which results in loss of in-memory search quality. It then performs a re-ranking using full-precision coordinates stored on SSD, using block-level data accesses but with expensive SSD accessing time. While methods such as DiskANN show promising results, the emergence of Heterogeneous Memory (HM) brings opportunities to significantly improve ANNS. HM combines cheap, slow but extremely large memory with expensive, fast but small memory (e.g., traditional DRAM) to achieve a good balance between production cost, memory performance and capacity. Because of the large memory capacity, HM can use full-precision vectors with accurate distance computation. Since memory access latency/bandwidth of slow memory components in HM is much faster than slow storage such as SSD, it is possible to occasionally access data in slow memory during search without paying the expensive cost of data accesses. That being said, realizing the full performance potential of HM for ANNS is still quite challenging. Although slow memory such as PMM performs $\sim$80X times faster than SSD, it is still $\sim$3X slower than DRAM in terms of random access latency [60]. Therefore, a naive data placement strategy can hurt the search efficiency badly. Therefore, one may still wonder if we can leverage HM for ANNS to achieve both high search accuracy and low search latency, especially when the dataset cannot fit in DRAM (fast memory)?

In this work, we revisit the similarity search problem in light of the recent advances in the field. Two new system optimization methods are introduced, dedicated to improving the efficiency and scaling of vector search while simultaneously delivering high accuracy. They are particularly appropriate for the new hardware architectures discussed above. Specially:

- iQAN [48] is a parallel search algorithm that exploits intra-query parallelism in graph-based vector search to obtain significant latency reduction in vector search on multi-core architectures with high accuracy.

This approach includes a set of optimizations that boost convergence, avoid redundant computations, and mitigate synchronization overhead.

- HM-ANN [51] is a heterogeneous memory-based technique that shatters the memory barrier of deploying large-scale vector search via NVMe memory, enabling billion-scale vector search with low deployment cost. It carefully constructs vector search indices via a memory hierarchy-aware algorithm, hence leading to substantially better search performance as the vectors grow to be larger than the DRAM capacity. It also employs parallel search algorithms to boost the in-memory search efficiency, leading to faster search speed.

In drawing broader lessons from this work, we believe that effectively leveraging multi-core and exploiting the memory hierarchy are the keys to high-performance vector search on modern processors. Further, based on the above methods, we discuss open research problems, including exploring hierarchical parallelism to meet both latency and throughput targets, highly concurrent vector search with addition and deletion, automating the index construction for vector search, and the interactions with modern applications in Section 5.

2 Background

The literature on nearest neighbor search is vast, and hence, we focus our attention on the most relevant works here. There has been a lot of work on building effective ANN indices to accelerate the search process. Earlier works focus on space partitioning-based methods. For example, Tree-based methods (e.g., KD-tree [53] and R* tree [8]) hierarchically split the data space into lots of regions that correspond to the leaves of a tree structure and only search a limited number of promising regions. However, the complexity of these methods becomes no more efficient than brute-force search as the dimension becomes large (e.g., $>$16) [35]. Prior works also have spent extensive efforts on locality-sensitive hashing-based methods [1, 2, 18, 29], which map data points into multiple buckets with a certain hash function such that the collision probability of nearby points is higher than the probability of others. These methods have solid theoretical foundations. LSH and its variations are often designed for large sparse vectors with hundreds of thousands of dimensions. In practice, LSH-based methods have been outperformed by other methods, such as graph-based approaches, by a large margin on large-scale datasets [4, 24, 40]. More recently, Malkov and Yashunin found graphs that satisfy the small-world property exhibit excellent navigability in finding nearest neighbors. They introduce the Hierarchical Navigable Small World (HNSW) [40], which builds a hierarchical k-NN graph with additional long-range links that help create the small-world property. For each query, it then performs a walk, which eventually converges to the nearest neighbor in logarithmic complexity. Subsequently, Fu et al. proposed NSG, which approximates Monotonic Relative Neighbor Graph (MRNG) [24] that also involves long-ranged links for enhancing connectivity.

3 iQAN: Fast and Accurate ANNS via Intra-Query Parallelism on Multi-Core Architecture

Among different vector search methods, the similarity graph-based algorithms have emerged as a remarkably effective class of methods for high-dimensional ANNS, outperforming other approaches on a wide range of datasets to achieve the best accuracy-vs-latency [4, 5, 20, 24, 31, 37, 59, 63]. Despite their promising results, graph-based methods still have challenges that limit their use in real-world scenarios. In particular, as the data size grows, it becomes increasingly challenging to achieve both low latency and high accuracy simultaneously. Existing solutions often resort to inter-query parallelism by dispatching queries across multiple processors or nodes to be processed simultaneously [7, 24]. This approach scales from a throughput perspective, but it does not help reduce query latency because each query still roughly performs the same amount of vector computations to find the nearest neighbors.

3.1 Challenges of ANNS via Intra-Query Parallelism

Another natural idea to reduce latency is to exploit intra-query parallelism on individual nodes with multi-core processors. For example, one may parallelize the node expansion in each iteration step of the sequential search algorithm (referred to as **Node-Expansion-in-Parallel**) because distance computations within a neighborhood expansion iteration do not have dependencies, hoping that multiple worker threads can check the closeness of multiple neighbors in parallel while performing the same computations on each step as the sequential algorithm. Surprisingly, this solution performs quite poorly and may even perform much worse than a well-tuned sequential algorithm, as shown in Fig. 1. There are several challenges in scaling ANNS with intra-query parallelism:

Challenge 1: Modern multi-core hardware is sensitive to synchronization overhead. Parallelism boosts compute capacity but may also incur high synchronization overhead, especially if there are complex data dependencies. While parallelizing the distance computation, Node-Expansion-in-Parallel also requires synchronization in between expansion iterations to sort the distance order of all candidates discovered by multiple parallel workers according to their distances to the query point, to decide which node to expand in the next iteration. We have observed that the synchronization is very expensive on a multi-core architecture, and frequent sequential-to-parallel synchronization as in Node-Expansion-in-Parallel can significantly prolong the search process. Fig. 2 shows that as we increase the number of threads, the synchronization overhead accounts for more than 50% of the total search time, becoming a dominating factor in the overall search latency.

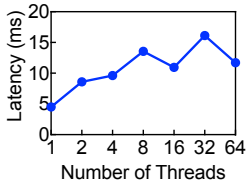


Figure 1: EP's latency on Deep100M.

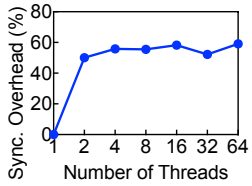


Figure 2: EP adds high sync. overhead.

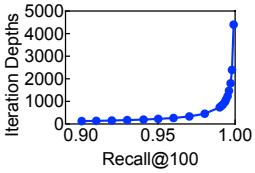


Figure 3: Iteration depths change along with recall.

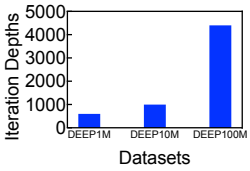


Figure 4: Iteration depths change along with data sizes.

Challenge 2: Node-Expansion-in-Parallel leads to insufficient computation granularity per worker, leading to sub-optimal memory bandwidth utilization. Node-Expansion-in-Parallel has low compute intensity because (1) unlike matrix multiplication, the point-wise Euclidean distance computation is an operator with low compute intensity, and (2) the number of neighbors to be expanded in one step is limited, given that similarity graphs naturally have low out-degree to avoid the out-degree explosion problem [24]. As such, further dividing the distance computation within each neighbor expansion iteration leads to insufficient work for each worker.

Challenge 3: Vector search using graph traversal requires many iterations to converge, resulting in long sequential dependencies between iterations and thus limiting its scalability. The number of neighborhood expansion iterations depends on the recall target and the graph size. For example, Fig. 3 shows that as the recall target increases, the number of iterations to find the top-100 nearest neighbors on a hundred million scale dataset DEEP100M grows dramatically as the recall target becomes higher (e.g., a 34.6-time increase from 0.9 to 0.999 recall). Fig. 4 shows that as the dataset size increases, the number of iterations to find the results for recall target 0.999 also grows (e.g., 7.3 times from 1M-vector dataset to 100M-vector dataset). This long sequential dependency makes achieving low latency with high accuracy especially challenging.

3.2 Design of iQAN

To address the aforementioned challenges, we introduce iQAN, a parallel search algorithm to accelerate graph-based ANNS on multi-core architectures with three key optimizations: (i) reducing neighbor expansion iteration

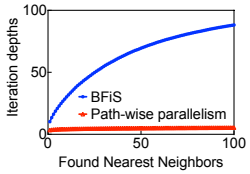


Figure 5: Iteration depths to find the K-th nearest neighbor (x-axis).

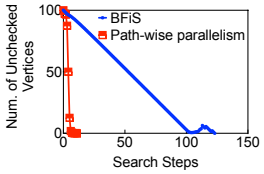


Figure 6: The number of steps for a search to converge.

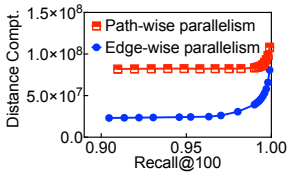


Figure 7: Aggregated distance computations of BFiS w/ EP and PP, where $W = 64$.

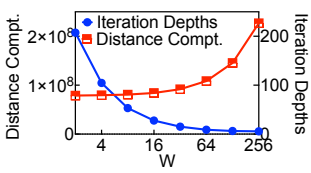


Figure 8: Dist. compt. increases as iter. depths decrease for PP increasing W.

depth by path-wise parallelism, (ii) reducing redundant distance computation by staged expansion, and (iii) reducing synchronization overhead by redundancy-aware synchronization.

3.2.1 Reduce Iteration Depth by Intra-Query Path-Wise Parallelism

In each search iteration, a Best-First-Search (BFiS) algorithm is often used to perform node expansion to the most promising unchecked candidate [24, 40]. In iQAN, we make a small modification to this process by relaxing the priority order and letting each thread expand a few more nodes (e.g., top W unchecked candidates) in every step as active nodes for expansion. We also relax the synchronization such that a global synchronization is only performed after a few expansion steps. We call this new way of expanding nodes path-wise parallelism (PP). This small change in algorithm results in a significant reduction in iteration depths for queries, e.g., from a few thousands to tens in some cases.

Why would this change reduce the iteration depth? The multi-node expansion and relaxed synchronizations are equivalent to letting each thread explore paths in a local region instead of a single node's neighbor list before doing a global synchronization. By doing so, it increases the likelihood of finding nearest neighbors in less number of iterations. Fig. 5 shows the comparison results of iteration depths between BFiS and PP on dataset SIFT1M using 10K queries with a 0.90 recall target. We set W to 64. Overall, while BFiS takes 10.1, 69.4, and 88.1 steps to find the top-1, top-50, and top-100 near neighbor, PP only takes 3.4, 5.0, and 5.4 steps on average, respectively, a significant reduction. From the unchecked node's perspective, Fig. 6 shows that PP also takes much fewer steps to converge to a local optimum (i.e., finish examining all the unchecked vertices) than BFiS.

3.2.2 Reduce Redundant Computation by Staged Expansion

Although reducing the iteration depth significantly, does it mean the search process will now get desired speedups on multi-core architectures? The answer is no. The path-wise parallelism reduces iteration depths but at the same time introduces a considerate amount of additional distance computations, especially when the number of parallel workers is large. Fig. 7 shows that to reach the same recall (0.9–0.999), the path-wise parallelism often needs to perform significantly more distance computations than BFiS (1.3–3.5 times). Moreover, we also observe that although the iteration depths continue to decrease by increasing the concurrent expansion width W, the number of distance computations inversely increases, as shown in Fig. 8. The huge amount of redundant computations adversely affects search efficiency as many threads are loading vectors for unnecessary computations, wasting memory bandwidth and compute resources.

To mitigate it, we investigate the usefulness of path-wise parallelism at different search stages: at which stage does the path-wise parallelism reduce the iteration depths the most? We found that overall, in the beginning, since all candidates are far from the query, those early expanded candidates are likely to be discarded by closer ones that are visited later. In other words, candidates expanded and checked at an earlier stage have a high likelihood of becoming unnecessary from a future perspective. As the search moves forward toward the region that has near

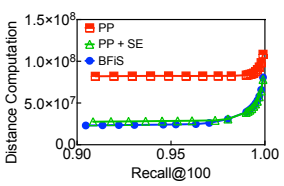


Figure 9: Dist. computation of BFiS, PP w/o and w/ staged expansion.

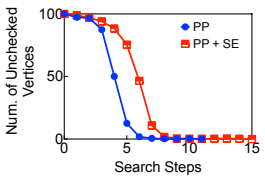


Figure 10: Number of unchecked candidates after each search step.

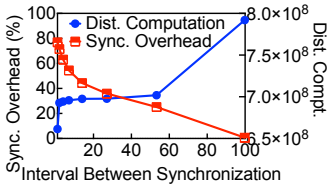


Figure 11: Sync. overhead and distance compt. as the sync. interval increases.

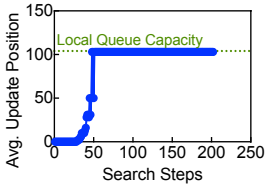


Figure 12: A query's average update positions during searching.

neighbors, a larger expansion width that covers more search paths can effectively prevent the search from getting stuck at a local minimum.

Based on these observations, we propose a staged expansion (SE) scheme by gradually increasing the expansion width W and the number of workers every t steps during the search procedure. In practice, we set the starting value of W to 1 and the maximum value as the number of available hardware threads. Then for every t steps (e.g., $t = 1$) we double the value of W until W reaches its maximum. Fig. 9 shows the comparison results of path-wise parallelism without and with staged expansion. The staged expansion reduces the number of redundant distance computations significantly, leading to distance computations comparable to BFiS. On the other hand, staged expansion is able to preserve the benefits of path-wise parallelism in terms of obtaining reduced iteration depths, as shown in Fig. 10. These results indicate that by performing path-wise parallelism at where they are most effective (i.e., the later phase of the search), the parallel search process can effectively converge with reduced iteration depths and minimal addition of redundant computations among multiple workers.

3.2.3 Reduce Synchronization Overhead by Redundancy-Aware Synchronization

The remaining performance challenge in parallel search resides in the synchronization, as we still need to decide when to do synchronization. However, reducing the synchronization overhead for graph-based ANNS is non-trivial. Fig. 11 shows that as we skip synchronizations in between search iterations (i.e., increasing the interval between two synchronizations), the synchronization overhead (shown as the ratio to the total time) decreases significantly. However, decreasing synchronization increases distance computations, especially when the synchronization intervals become large. This is because as we increase the synchronization interval, it increases the likelihood that individual workers would search their local but unpromising areas without switching to newly identified promising regions found by other workers. As such, one cannot infinitely delay synchronization, and a small set but useful synchronizations are desired to achieve overall high search efficiency without incurring too many redundant computations.

Finding such intervals turns out to be non-trivial since the relative distance of a query to its near neighbors changes all the time at different stages. It is also hard to find one fixed synchronization interval for all queries. To mitigate the synchronization overhead, iQAN performs redundancy-aware synchronization (RAS), which allows workers to perform a search with low redundant computations by adding a minimal set of global synchronizations. We introduce a metric — update positions — to capture the redundancy during expansion. When a worker thread expands an unchecked candidate, its unchecked neighbors are then inserted into the worker's local queue, and we define the update position as the lowest (best) position of all newly inserted candidates. Thus, the average update position (AUP) is the mean of all update positions of workers. Fig. 12 demonstrates how an example query's AUP changes during the search process without doing any global synchronizations. We observe that the AUP increases gradually to be equal to the local queue capacity and remains flat to the end. When the AUP is close to the queue capacity, it indicates that a majority of workers are searching areas that cannot find promising candidates to update their local results. Therefore, a high AUP indicates that most workers are doing redundant

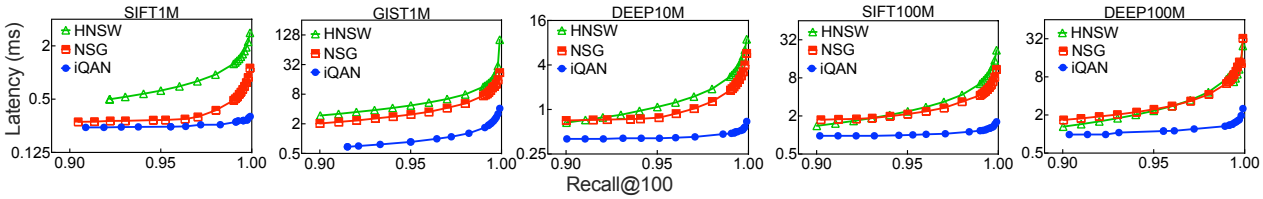


Figure 13: Latency comparison among HNSW, NSG, and iQAN on Skylake (16T).

computations, and it would benefit from a global synchronization such that all workers can focus on searching for more promising areas that have a higher probability of including closer near neighbors.

3.3 Evaluation of iQAN

iQAN offers significant speedups than two state-of-the-art graph-based ANNS, NSG [23, 24] and HNSW [39, 40] over several public datasets, including SIFT1M (128D), GIST1M (960D), DEEP10M (96D), DEEP100M (96D), and SIFT100M (128D). We measure the latency and Recall@100 (R@100), which measures the accuracy of finding the top-100 nearest neighbors for every query. We conduct our experiments on a workstation with Xeon Gold 6138 (2.00 GHz) with 20 cores and 128 GB DRAM (Skylake for short).

Fig. 13 compares the latency of HNSW, NSG, and iQAN on Skylake. NSG and HNSW use their sequential search algorithm, whereas iQAN uses 16 threads on Skylake. Across all five datasets, iQAN consistently provides latency speedups over existing sequential-based approaches NSG and HNSW over a wide range of recall targets. In particular, the speedups from iQAN increase as the recall target moves to the high accuracy regime (e.g., from 0.90 to 0.999). Notably, iQAN achieves up to 12.9× speedups over NSG on DEEP100M on Skylake, obtaining an incredibly low latency of <5ms or <3ms at the recall target 0.999 by leveraging aggregated multi-core computation and memory bandwidth resources. This enables vector search with very high accuracy on large-scale graphs, even in extremely interactive online applications.

iQAN achieves significant latency speedups mainly for three reasons. First, iQAN's path-wise parallelism effectively reduces the iteration depths, making the sequential dependencies no longer a major bottleneck. This is particularly critical for a large graph (e.g., DEEP100M) and high recall (e.g., 0.999) as seen in Section 3.2 that the iteration depths increase significantly as we either scale the graph size or increase the recall targets. Second, the reduced iteration depths do not come at the cost of many redundant computations as iQAN leverages staged expansion to effectively avoid redundant computations from doing path-wise parallelism. Third, iQAN significantly reduces the synchronization overhead through redundancy-aware synchronization. It is also worth mentioning that iQAN achieves excellent speedups as we increase the dimensionality of the embedding vectors. iQAN achieves up to 24.9× speedups over HNSW on GIST1M on Skylake. This is higher than the speedups we get on a dataset with a similar scale but much smaller dimensionality (e.g., SIFT1M). iQAN is able to achieve better speedups on higher dimensional vectors because as the vector dimension increases, the amount of computation workload for the pair-wise distance computation also increases, which allows iQAN to benefit more from parallel computing.

Comparison with DiskANN. Fig. 14 compares the latency of DiskANN [30] (using 1 thread with its in-memory index) and iQAN (using 32 threads) for Recall@1 targets. For building its indices of datasets SIFT1M and GIST1M, DiskANN uses $L = 125$, $R = 70$, $\alpha = 2$, which are the same setting as shown in its paper. For DEEP10M, DiskANN uses $L = 100$, $R = 100$, $\alpha = 1.2$. Fig. 14 shows that iQAN achieves significant latency speedups over DiskANN, especially for the high recall regime. For example, for recall target 0.999, iQAN has about 180.5× average speedup on DiskANN among these three datasets.

Scaling to billion points. This experiment is conducted on a machine with a 1.5 TB memory. It is worth mentioning that even 1.5 TB of memory is not enough to build a 100-NN graph with one billion data vectors.

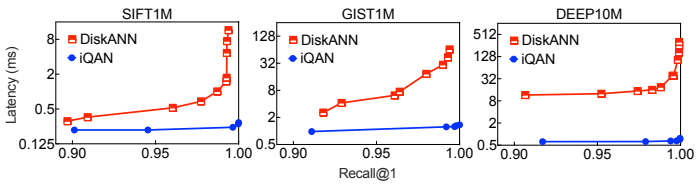


Figure 14: Recall@1 latency of DiskANN and iQAN.

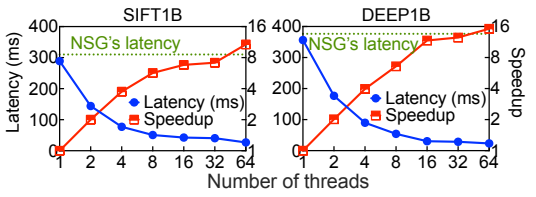


Figure 15: Performance comparison of iQAN and NSG on two billion-scale datasets SIFT1B (`bigann`) and DEEP1B.

Therefore, we limit the out-degree of NSG when generating the corresponding NSG index so that the index construction can finish in a reasonable amount of time (e.g.,<10 days). We also note that this is the first time to evaluate an NSG graph at a billion scale as the maximum graph prior work such as NSG evaluated contained less than 100M data points. Fig. 15 compares the latency of iQAN and NSG. iQAN uses up to 64 threads, and the recall target is 0.9. When using 64 threads, iQAN follows the trend of scalability we observed as we increase the graph size and outperform NSG with $11.5\times$ and $16.0\times$ speedup for SIFT1B and DEEP1B, respectively, confirming the excellent scalability of iQAN on large-scale graphs again.

4 HM-ANN: Efficient Billion-Point Vector Search via Heterogeneous Memory

This section describes a large-scale vector search solution built on top of Heterogeneous Memory. The HNSW index is modified to make it HM-aware, and the search algorithm is also modified to make the search more efficient. With these modifications, this solution enables fast and highly accurate billion-scale ANNS on HM.

4.1 Design of HM-ANN

The design of HM-ANN generalizes HNSW, whose hierarchical structure naturally fits into HM. Elements in the upper layers consume a small portion of the memory, making them good candidates to be placed in fast memory (small capacity); The bottom-most layer has all the elements and has the largest memory consumption, which makes it suitable to be placed in slow memory. Unlike HNSW, where the majority of search happens in the bottom-most layer, elements in the upper layers now have faster access speed, so it is a reasonable strategy to increase the access frequency of the upper layers. On the other hand, since accessing L0 is slower, it is preferable to have only a small portion of it to be accessed by each query. The key idea of HM-ANN is, therefore, to build high-quality upper layers and make most memory accesses happen in fast memory, in order to provide better navigation for search at L0 and reduce memory accesses in slow memory.

Notations. In the rest of the paper, we let V denote the dataset with $N = |V|$ to build the graph; we refer the graph in the layer $i \in \{0, 1, ..., l\}$ of HM-ANN as $G_i = (V_i, E_i)$ where V_i is the vertex set and E_i is the edge set. We refer N_i as the number of elements in the layer i, and we have $N_i = |V_i|$. Because L0 contains all the elements in the database, we have $V_0 = V$ and $N_0 = N$. Based on the hierarchical structure of HM-ANN, we have $V_i \subsetneq V_{i-1}$. Similar to the existing effort [40], we introduce M_i as the maximum number of established connections for each point v in the layer i. For $v \in V$, we let $D(v)$ denote the degree of node v, and $D(v) = \sum_{u \in V} m(v, u)$ where $m(v, u) = 1$ if there exists a link between node v and node u.

4.1.1 HM-aware Index Construction via Top-Down Insertions and Bottom-up Promotions

To make the ANNS index aware of HM architecture, we generalize the HNSW construction algorithm to include two phases: a top-down insertion phase and a bottom-up promotion phase.

Top-down insertions. The top-down insertion phase is the same as HNSW, where we incrementally build a hierarchical graph by iteratively inserting each vector v in V as a node in G. Each node will generate up to M (i.e., the neighbor degree) out-going edges. Among those, $M-1$ are short-range edges, which connect v to its $M-1$ nearest neighbors according to their pair-wise Euclidean distance to v. The rest is a long-range edge that connects v to a randomly picked node, which may connect other isolated clusters. It is theoretically justified that graphs (e.g., L0) constructed by inserting these two types of edges guarantee to have the small world properties [24, 40, 61].

Bottom-up promotions. The goal of the second phase is to build a high-quality projection of L0 elements into the layer 1 (L1), such that the search in L0 can find the true nearest neighbors of the query with only a few hops. Ideally, HM-ANN wants to achieve the goal that performing 1-greedy search in L0 is sufficient to achieve high recall, so that the slowdown caused by accessing the slow memory is minimal. A straightforward way to project the L0 elements into L1 is to randomly select a subset of elements in L0 to be L1, similar to what HNSW already does to build upper layers. However, we observe that such an approach leads to poor index quality. As a result, many searches end up happening in L0 (slow memory), causing long search latency.

HM-ANN uses a *high-degree promotion strategy*. This strategy promotes elements with the highest degree in L0 into L1. From the layer i ($i \geq 2$) to $i+1$, HM-ANN promotes high-degree nodes to the upper layer with a promotion rate of $1/M$, where M is the maximum number of neighbors for each element (i.e., $M_i = M$, where $i = 2...l$). A similar promotion rate setting is used in HNSW [40] and typical skip list [50]. HM-ANN increases search quality in $L1$ by promoting more nodes from $L0$ to $L1$ and setting the maximum number of neighbors for each element in L1 to $2 \times M$ (i.e., $M_1 = 2 \times M$). The number of nodes in upper layers (N_i, where $i = 1..l$) is decided by available fast memory space.

The high-degree promotion strategy is based on the following observation. The hub nodes of the graph at L0 are those nodes with a large number of connections (i.e., high degree). In the small world navigation algorithm, a higher degree node provides better navigability [6]. Most of the shortest paths between nodes flow through hubs. In other words, the average length of the navigation path (i.e., number of hops) is the smallest, when the adjacent node with the highest degree is selected as the next hop. By promoting the high-degree nodes, the resulting L1 layer allows HM-ANN to effectively reduce the number of search steps in L0, compared with the random promotion strategy.

4.1.2 HM-ANN Graph Search Algorithm

Fast memory search. The search in fast memory begins at the entry point in the top layer and then performs a 1-greedy search from the top layer to layer 2, which is the same as in HNSW. To narrow down the search space in L0, HM-ANN performs the search in L1 with a search budget controlled by $efSearch_{L1}$. $efSearch_{L1}$ defines the size of dynamic candidate list in L1. Those candidates in the list are used as entry points for search in L0 (HNSW uses just one entry point), in order to improve search quality in L0.

Parallel L0 search. In L0, HM-ANN evenly partitions the candidates from searching L1 and uses them as entry points to perform parallel multi-start 1-greedy search with Thr threads in parallel. The top candidates from each search are collected to find the best candidates. Parallel search makes the best use of memory bandwidth and improves search quality without increasing search time. Thr is determined by peak memory bandwidth constrained by hardware divided by memory bandwidth consumption by one thread, which is easy to calculate.

Different from the SSD-based ANNS [55, 66], the data in slow memory in HM-ANN can be directly accessed by processors, and there is no duplication between fast and slow memories. However, due to high latency and low bandwidth of slow memory, HM-ANN should still make memory accesses in fast memory as many as possible. HM-ANN implements a software-managed cache in fast memory to prefetch data from slow memory to fast memory before the memory access happens. In particular, HM-ANN reserves a space in fast memory ($\sim$2 GB) called *migration space*. When searching L1, HM-ANN asynchronously copies neighbor elements of those candidates in $efSearch_{L1}$ and the neighbor elements' connections in L1 from slow memory to the migration

Table 1: Indexing time and memory consumption for graph-based methods on billion-scale datasets

	BigANN					DEEP1B				
	Indexing			Search		Indexing			Search	
	Graph size	Indexing time	Promo. rate	Fast-mem usage	Slow-mem usage	Graph size	Indexing time	Promo. rate	Fast-mem usage	Slow-mem usage
HNSW	475GB	90h	0.02	96GB (hw caching)	490GB	723GB	108h	0.02	96GB (hw caching)	748GB
NSG	285GB	115h	-	96GB (hw caching)	303GB	580GB	134h	-	96GB (hw caching)	599GB
HM-ANN	536GB	96h	0.16	96GB	462GB	756GB	117h	0.11	96GB	681GB

space in fast memory. When the search in L0 happens, there is already a portion of to-be-accessed data placed in fast memory, which leads to shorter query time.

4.2 Evaluation of HM-ANN

Billion-scale algorithm comparison. We compare HM-ANN with the graph- (HNSW and NSG) and quantization-based algorithms (IMI+OPQ and L&C). For HNSW, we build graphs with $efConstruction$ and M set to 200 and 48 respectively; For NSG we first build a 100-NN graph using Faiss [42] and then build NSG graphs with R = 128, L = 70 and C = 500. We collect results on NSG and HNSW using Memory Mode since it leads to overall better performance than using first-touch NUMA. For IMI+OPQ, we build indexes with 64- and 80-byte code books on BIGANN and DEEP1B respectively. We present the best search result with search parameters nprobe=128 and ht=30 for BIGANN and with autotuning parameter sweep on DEEP1B. For L&C, we use 6 as the number of links on the base level, and use 36- and 144-byte OPQ code-books. We use the same parameters ($efConstruction$=200 and M=48) as HNSW to construct HM-ANN. We set $efSearch_{L0}$=2 and vary $efSearch_{L1}$ to show the latency-vs-recall trade-offs.

Figures 16 (a)-(d) visualize the results. Overall, HM-ANN provides the best latency-vs-recall performance. Figure 16 (a) and (b) show that HM-ANN achieves the top-1 recall of $> 95\%$ within 1ms, which is 2x and 5.8x faster than HNSW and NSG to achieve the same recall target respectively. IMI+OPQ and L&C cannot reach a similar recall target, because of precision loss from quantization. As another point of reference, the SSD-based solution, DiskANN [55] (not open-sourced), provides 95% top-1 recall in 3.5ms. In contrast, HM-ANN provides the same recall in less than 1ms, which is at least 3.5$\times$ faster. We compare the top-100 recall shown in Figures 16 (c) and (d). HM-ANN provides higher performance than all other approaches. For example, it obtains top-100 recall of $> 90\%$ within 4 ms, while performing 2.8x and 5x faster than HNSW and NSG with the same recall target respectively. Quantization-based algorithms perform poorly and have difficulties to reach a top-100 recall of 30%.

Table 1 shows the index construction time and index size of HNSW, NSG, and HM-ANN. Among the three, HNSW takes the shortest time to build the graph. HM-ANN takes 8% longer time than HNSW because it takes an additional pass for the bottom-up promotion. However, HM-ANN is still faster to construct than NSG. In terms of memory usage, HM-ANN indexes are 5–13% larger than HSNW, because it promotes more nodes from L0 to L1. In terms of memory usage, HM-ANN consumes less fast memory than HNSW and NSG, which is valuable to reduce production cost [41, 52]. HNSW and NSG use all fast memory because they do not explicitly manage HM and by default using Memory Mode consumes all fast memory. The sum of slow and fast memory consumption can be larger than the index size because there are metadata needed for search that are not counted in the index size.

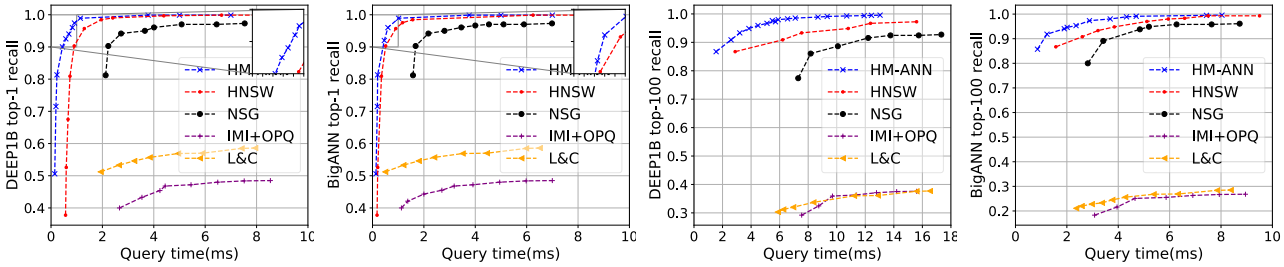


Figure 16: Query time vs. recall curve in (a) DEEP1B top-1, (b) BigANN top-1, (c) DEEP1B top-100, (b) BigANN top-100, respectively.

5 Research Opportunities

In this paper, we have concentrated on the computational and memory efficiency of large-scale vector search. Our iQAN and HM-ANN designs achieve exceptional performance. However, large-scale vector search still requires further attention, and we expect that additional improvements can be made. This section explores the challenges and research opportunities associated with vector search.

High-performance vector search through hierarchical parallelism. Inference demand varies across different applications and scenarios, with some being latency-critical, and others being latency-sensitive or throughput-oriented. Additionally, the hardware resources available to each application may also vary significantly. To meet these diverse requirements and make efficient use of all computational resources, we can combine different parallelism approaches:

- Distributed vector search. Libraries such as Milvus [57] allow the development of distributed versions of vector search systems using a cluster.
- Inter-query parallelism. Multithreading enables the use of multiple cores to improve the throughput of answering user requests.
- Intra-query parallelism. Methods such as iQAN enable the system to speed up individual queries by leveraging the aggregated computational capacity of multiple cores.

These techniques can be combined hierarchically to meet given latency, throughput, and cost objectives. The research opportunity here is: Build systems for large-scale vector search that fully exploit parallelism at different levels, providing millisecond-level latency, high throughput, and low hardware cost. Furthermore, while techniques such as iQAN optimize search efficiency by modifying the graph traversal process to leverage intra-query parallelism, the underlying index has not been specifically designed to take advantage of intra-query parallelism. Intuitively, a different index structure could achieve better search efficiency under intra-query parallelism if it naturally helps avoid redundant computations across different threads and also leads to better data locality. The research opportunity here is: Design vector search indices that maximize search efficiency through both inter- and intra-query parallelism.

Highly concurrent vector search with addition and deletion. Most existing vector search systems build indices offline and become read-only once deployed. In some applications, data capture may occur more frequently than query processing, or the application may need to index continuous data streams while serving query requests. In these cases, the index organization should be optimized for addition and update in addition to query performance. The complexity arises with concurrent read and update operations (e.g., addition, deletion), as it is challenging to achieve both correctness and speed simultaneously. Concurrent updates and reads can lead to data races, making it difficult to ensure correctness on shared-memory architectures. Lock-based synchronization

can be used to coordinate access to shared-memory data, but while using one lock for the entire index simplifies reasoning about correctness, it also leads to serialized execution, severely impacting scalability. Another solution is to use fine-grained locking, where multiple locks are associated with the index. However, fine-grained locking increases the complexity of operations that access shared data, leading to issues such as deadlock, atomicity violation, and high locking overhead. The research opportunity here is: Build a highly concurrent vector search system that supports robust addition and deletion with high search efficiency.

Automating index construction for vector search. Several advances have been made in support of large-scale vector search, introducing novel algorithms and system optimizations [15, 51, 55, 66]. However, applying these methods to large-scale datasets still requires a significant amount of engineering effort specific to the data, hardware environment, and performance objectives. For instance, deploying a large dataset to a given cloud VM requires a careful selection of vector search indices, with NVMe/SSD/remote storage, the number of cores to use, and multiple index-specific parameters. Correctly choosing the algorithm and tuning the parameters can deliver significant improvements in search performance but also depend on strong system expertise. Therefore, automating the selection and construction of vector search indices would help alleviate the burden on deployment engineers, but it also requires navigating a complex space of choices that grows exponentially with different algorithm choices, each with its own trade-offs, and data sizes and hardware resources. The research opportunity here is: Build a framework and optimization algorithm that automatically finds the optimal index construction strategy for a given dataset to deliver fast search speed with low cost.

6 Conclusion

We have designed and implemented iQAN and HM-ANN to enhance the system efficiency of vector search. We have followed the state-of-the-art vector search algorithm, but at every level, we have introduced innovations that stretch prior methods and tailor our system for the newer hardware setting of multi-core processors and heterogeneous memory. Our innovations, such as intra-query path-wise parallelism, staged expansion, redundancy-aware synchronization, and HM-aware index construction, improve the computational and memory efficiency of large-scale vector search. Many of these methods should work well in other settings, such as SSD-based methods and compression-based indices. Finally, we propose several open research directions from a system perspective that have the potential to further improve large-scale vector search. We hope these directions will inspire new research that can advance vector search and make it more efficient, robust, and easy-to-use.

References

[1] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In 2006 47th annual IEEE symposium on foundations of computer science (FOCS'06), pages 459–468. IEEE, 2006. doi: 10.1109/FOCS.2006.49.

[2] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS), volume 28 of NIPS'15, pages 1225–1233, Cambridge, MA, USA, 2015. MIT Press. URL https://proceedings.neurips.cc/paper/2015/file/2823f4797102ce1a1aec05359cc16dd9-Paper.pdf.

[3] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate Nearest Neighbor Search in High Dimensions. arXiv preprint arXiv:1806.09823, 2018.

[4] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In International Conference on Similarity Search and Applications (SISAP), pages 34–49. Springer, 2017.

[5] Artem Babenko and Victor Lempitsky. The inverted multi-index. IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(6):1247–1260, 2014.

[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. Science, 286(5439):509–512, 1999. ISSN 0036-8075. doi: 10.1126/science.286.5439.509. URL https://science.sciencemag.org/content/286/5439/509.

[7] KG Renga Bashyam and Sathish Vadhiyar. Fast scalable approximate nearest neighbor search for high-dimensional data. In 2020 IEEE International Conference on Cluster Computing (CLUSTER), pages 294–302. IEEE, 2020. doi: 10.1109/CLUSTER49012.2020.00040.

[8] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In SIGMOD 1990, pages 322–331, 1990.

[9] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, 18(9):509–517, September 1975. ISSN 0001-0782.

[10] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. Nature Biotechnology, 33 (6):623–630, 2015.

[11] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When Is "Nearest Neighbor" Meaningful? In International Conference on Database Theory (ICDT), pages 217–235, 1999.

[12] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in High-Dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases. ACM Computing Surveys, 33(3):322–373, 2001.

[13] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA, volume 162 of Proceedings of Machine Learning Research, pages 2206–2240. PMLR, 2022.

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

[15] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. SPANN: highly-efficient billion-scale approximate nearest neighborhood search. In Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, pages 5199–5212, 2021.

[16] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Shilad Sen, Werner Geyer, Jill Freyne, and Pablo Castells, editors, Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016, pages 191–198. ACM, 2016.

[17] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: Scalable online collaborative filtering. In Proceedings of the 16th International Conference on World Wide Web (WWW), pages 271–280, 2007.

[18] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG '04, pages 253–262, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138857. doi: 10.1145/997817.997857. URL https://doi.org/10.1145/997817.997857.

[19] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. Link and code: Fast indexing with graphs and compact regression codes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3646–3654, 2018.

[20] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. Proceedings of the VLDB Endowment, 13(3): 403–420, 2019.

[21] Nanyi Fei, Zhiwu Lu, Yizhao Gao, Guoxing Yang, Yuqi Huo, Jingyuan Wen, Haoyu Lu, Ruihua Song, Xin Gao, Tao Xiang, et al. Towards artificial general intelligence via a multimodal foundation model. Nature Communications, 13 (1):3094, 2022.

[22] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing Web Latency: The Virtue of Gentle Aggression. In Proceedings of the ACM Conference of the Special Interest Group on Data Communication, SIGCOMM '13, pages 159–170, 2013. ISBN 978-1-4503-2056-6. doi: 10.1145/2486001.2486014. URL http://doi.acm.org/10.1145/2486001.2486014.

[23] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Nsg. https://github.com/ZJULearning/nsg, 2019. [Online; accessed 25-April-2022].

[24] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. Proceedings of the VLDB Endowment (VLDB), 12(5):461–474, January 2019. ISSN 2150-8097. doi: 10.14778/3303753.3303754. URL https://doi.org/10.14778/3303753.3303754.

[25] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In 2013 IEEE Conference on Computer Vision and Pattern Recognition, pages 2946–2953, 2013.

[26] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In VLDB'99, pages 518–529, 1999.

[27] Johannes Hoffart, Stephan Seufert, Dat Ba Nguyen, Martin Theobald, and Gerhard Weikum. Kore: Keyphrase overlap relatedness for entity disambiguation. In Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM), pages 545–554, 2012.

[28] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In CIKM '13, pages 2333–2338, 2013.

[29] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC '98, pages 604–613, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919629. doi: 10.1145/276698.276876. URL https://doi.org/10.1145/276698.276876.

[30] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32, pages 13771–13781. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf.

[31] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(1):117–128, 2011. doi: 10.1109/TPAMI.2010.57.

[32] Yannis Kalantidis and Yannis S. Avrithis. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In CVPR 2014, pages 2329–2336, 2014.

[33] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In 2009 IEEE 12th International Conference on Computer Vision (ICCV), pages 2130–2137. IEEE, 2009.

[34] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. In Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014, volume 32 of JMLR Workshop and Conference Proceedings, pages 1188–1196. JMLR.org, 2014.

[35] D. T. Lee and C. K. Wong. Worst-case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees. Acta Informatica, 9(1):23–29, March 1977.

[36] Victor Lempitsky. The inverted multi-index. In CVPR '12, pages 3069–3076, 2012.

[37] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data – experiments, analyses, and improvement. IEEE Transactions on Knowledge and Data Engineering, 32(8):1475–1488, 2020. doi: 10.1109/TKDE.2019.2909204.

[38] Qin Lv, Moses Charikar, and Kai Li. Image similarity search with compact data structures. In Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management (CIKM), pages 208–217, 2004.

[39] Yury A Malkov and Dmitry A Yashunin. Hnswlib. https://github.com/nmslib/hnswlib, 2020. [Online; accessed 25-April-2022].

[40] Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4):824–836, 2020. doi: 10.1109/TPAMI.2018.2889473.

[41] Jon Martindale. RAM has never been cheaper, but are the historic prices here to stay? . https://www.digitaltrends.com/computing/why-is-ram-so-cheap/, 2019.

[42] Meta. FAISS. `https://github.com/facebookresearch/faiss/tree/master/benchs`, 2017.

[43] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, 2013.

[44] Marius Muja and David G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. TPAMI 2014, 36(11):2227–2240, 2014.

[45] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian Allen Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. Semantic product search. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019, pages 2876–2885. ACM, 2019.

[46] Mohammad Norouzi and David J. Fleet. Cartesian K-Means. In CVPR 2013, 2013.

[47] Mohammad Norouzi, David J. Fleet, and Ruslan Salakhutdinov. Hamming distance metric learning. In Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, pages 1070–1078, 2012.

[48] Zhen Peng, Minjia Zhang, Kai Li, Ruoming Jin, and Bin Ren. iqan: Fast and accurate vector search with efficient intra-query parallelism on multi-core architectures. In Maryam Mehri Dehnavi, Milind Kulkarni, and Sriram Krishnamoorthy, editors, Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, PPoPP 2023, Montreal, QC, Canada, 25 February 2023 - 1 March 2023, pages 313–328. ACM, 2023.

[49] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In 2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–8. IEEE, 2007.

[50] William Pugh. Skip lists: A probabilistic alternative to balanced trees. In F. Dehne, J. R. Sack, and N. Santoro, editors, Algorithms and Data Structures, pages 437–449, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. ISBN 978-3-540-48237-6.

[51] Jie Ren, Minjia Zhang, and Dong Li. HM-ANN: efficient billion-point nearest neighbor search on heterogeneous memory. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[52] Gina Roos. Dram Prices Continue to Climb. https://epsnews.com/2017/08/18/dram-prices-continue-climb/, 2017.

[53] Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In 2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–8. IEEE, 2008.

[54] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[55] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Rand-nsg: Fast accurate billion-point nearest neighbor search on a single node. In NeurIPS, pages 13748–13758, 2019.

[56] Danny Sullivan. Faq: All about the google rankbrain algorithm. https://searchengineland.com/faq-all-about-the-new-google-rankbrain-algorithm-234440, 2018.

[57] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. Milvus: A purpose-built vector data management system. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, pages 2614–2627. ACM, 2021.

[58] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. arXiv preprint arXiv:2101.12631, 2021.

[59] Runhui Wang and Dong Deng. Deltapq: Lossless product quantization code compression for high dimensional similarity search. Proceedings of the VLDB Endowment, 13(13):3603–3616, 2020.

[60] Zixuan Wang, Xiao Liu, Jian Yang, Theodore Michailidis, Steven Swanson, and Jishen Zhao. Characterizing and modeling non-volatile memory systems. In 53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020, pages 496–508. IEEE, 2020.

[61] Duncan J. Watts. Small Worlds: The Dynamics of Networks Between Order and Randomness. Princeton University Press, 1999.

[62] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In Proceedings of the 24th VLDB Conference, pages 194–205, 1998.

[63] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. Proceedings of the VLDB Endowment, 13(12):3152–3165, 2020.

[64] Samuel Williams, Andrew Waterman, and David A. Patterson. Roofline: an insightful visual performance model for multicore architectures. In CACM'09, 2009.

[65] Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. Neural Ranking Models with Multiple Document Fields. In WSDM '18, 2018.

[66] Minjia Zhang and Yuxiong He. GRIP: multi-store capacity-optimized high-performance nearest neighbor search for vector search engine. In CIKM 2019, pages 1673–1682, 2019.

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.

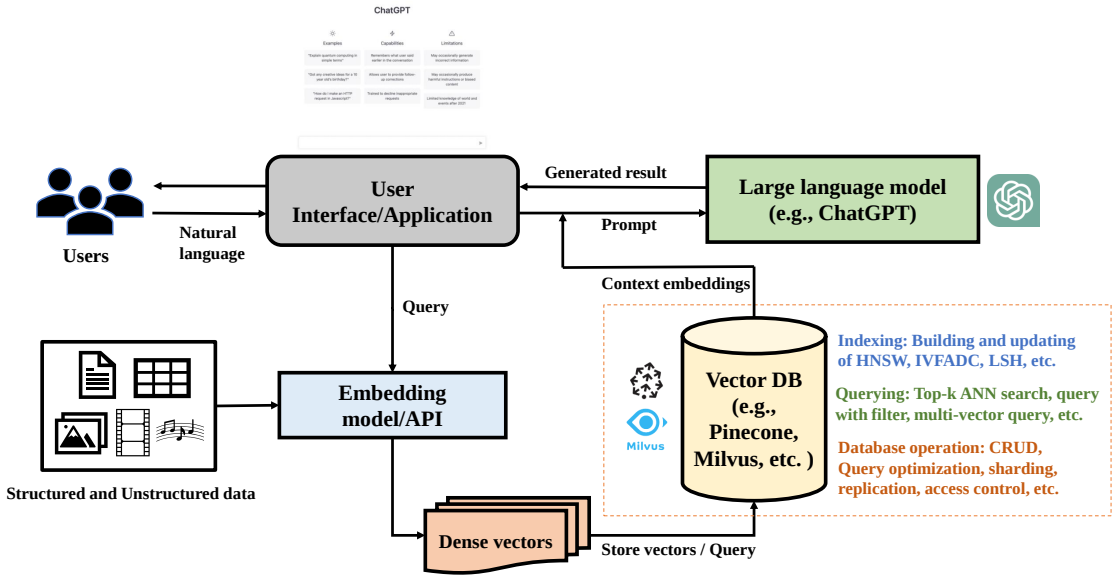


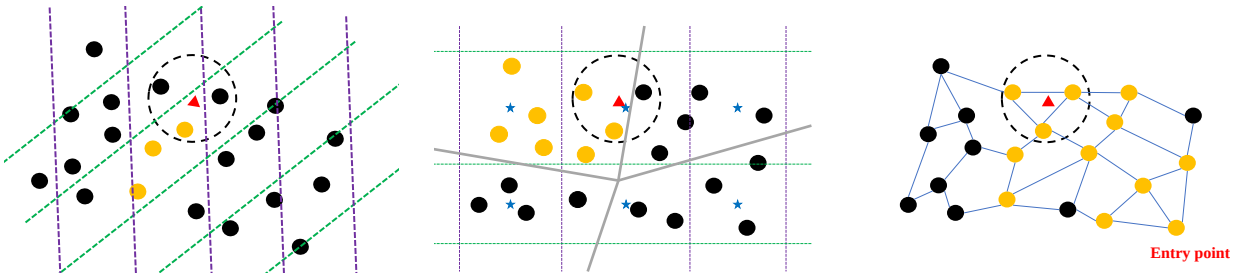
Figure 1: Framework for Enhancing Answer Quality in LLM-based Chatbots using Vector Databases

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)

Learning Space Partitions for Nearest Neighbor Search

Yihe Dong[†], Piotr Indyk[‡], Ilya Razenshteyn[§], Tal Wagner[‖]
[†]Work done at Microsoft Research. Now at Google.
[‡]MIT.
[§]Work done at Microsoft Research. Now at Pyte.
[‖]Work done at Microsoft Research and MIT. Now at Amazon and Tel-Aviv University.

Author names are ordered alphabetically.

Abstract

Space partitions of $\mathbb{R}^d$ underlie a vast and important class of fast nearest neighbor search (NNS) algorithms. Inspired by recent theoretical work on NNS for general metric spaces [8, 9], we develop a new framework for building space partitions reducing the problem to balanced graph partitioning followed by supervised classification. We instantiate this general approach with the KaHIP graph partitioner [50] and neural networks, respectively, to obtain a new partitioning procedure called Neural Locality-Sensitive Hashing (Neural LSH). On several standard benchmarks for NNS [3], our experiments show that the partitions obtained by Neural LSH consistently outperform partitions found by quantization-based and tree-based methods as well as classic, data-oblivious LSH.

1 Introduction

The *Nearest Neighbor Search (NNS)* problem is defined as follows. Given an n-point dataset P in a d-dimensional Euclidean space $\mathbb{R}^d$, we would like to preprocess P to answer k-nearest neighbor queries quickly. That is, given a query point $q \in \mathbb{R}^d$, we want to find the k data points from P that are closest to q. NNS is a cornerstone of the modern data analysis and, at the same time, a fundamental geometric data structure problem that led to many exciting theoretical developments over the past decades. See, e.g., [7, 53] for an overview.

The main two approaches to constructing efficient NNS data structures are indexing and sketching. The goal of indexing is to construct a data structure that, given a query point, produces a small subset of P (called candidate set) that includes the desired neighbors. Such a data structure can be stored on a single machine, or (if the data set is very large) distributed among multiple machines. In contrast, the goal of sketching is to compute compressed representations of points to enable computing approximate distances quickly (e.g., compact binary hash codes with the Hamming distance used as an estimator, see the surveys [53, 54]). Indexing and sketching can be (and often are) combined to maximize the overall performance [28, WGS+17].

Both indexing and sketching have been the topic of a vast amount of theoretical and empirical literature. In this work, we consider the indexing problem. In particular, we focus on indexing based on space partitions. The overarching idea is to build a partition of the ambient space $\mathbb{R}^d$ and split the dataset P accordingly. Given a query point q, we identify the bin containing q and form the resulting list of candidates from the data points residing in the same bin (or, to boost the accuracy, nearby bins as well).

Some of the popular space partitioning methods include the following:

- locality-sensitive hashing (LSH) [5, 6, 25, 37], where the partitioning is obtained by hashing the points into "bins" such that the probability of two points colliding dependends on the distance between them,

- quantization-based approaches [14, 29, GSL+20], where partitions are obtained via k-means clustering of the dataset and its generalizations, and
- tree-based methods, such as random-projection trees, learned trees or PCA trees [10, 17, 20, 33, 39, 49]. Here, the partition is defined by a rooted binary tree, where each internal node v is augmented with a hyperplane, and each edge to a child of v corresponds to one of the two halfspaces defined by the hyperplane. In turn, each leaf v in the tree corresponds to a cell in the partition, defined as the intersection of all halfspaces corresponding to the edges on a path from the root to v.

Compared to other indexing methods (see Section 1.2), space partitions have multiple benefits. First, they are naturally applicable in distributed settings, as different bins can be stored on different machines [12, 15, 36, 44]. Moverover, the computational efficiency of search can be further improved by using any nearest neighbor search algorithm locally on each machine. Second, partition-based indexing is particularly suitable for GPUs due to the simple and predictable memory access pattern [28]. Finally, partitions can be combined with cryptographic techniques to yield efficient secure similarity search algorithms [16]. Thus, in this paper we focus on designing space partitions that optimize the trade-off between their key metrics: the number of reported candidates, the fraction of the true nearest neighbors among the candidates, the number of bins, and the computational efficiency of the point location.

Recently, there has been a large body of work that studies how modern machine learning techniques (such as neural networks) can help tackle various classic algorithmic problems (a partial list includes [11, 13, 19, 32, 40–42, 45]). Similar methods—under the name "learn to hash"—have been used to improve the sketching approach to NNS [53]. However, when it comes to indexing, while some unsupervised techniques such as PCA or k-means have been successfully applied, the full power of modern tools like neural networks has not yet been harnessed. This state of affairs naturally leads to the following general question: **Can we employ modern (supervised) machine learning techniques to find good space partitions for nearest neighbor search?**

1.1 Our contribution

In this paper we address the aforementioned challenge and present a new framework for finding high-quality space partitions of $\mathbb{R}^d$. Our approach consists of three major steps:

1. Build the k-NN graph G of the dataset by connecting each data point to k nearest neighbors;
2. Find a balanced partition $\mathcal{P}$ of the graph G into m parts of nearly-equal size such that the number of edges between different parts is as small as possible;
3. Obtain a partition of $\mathbb{R}^d$ by training a classifier on the data points with labels being the parts of the partition $\mathcal{P}$ found in the second step.

See Figure 1 for illustration. The new algorithm directly optimizes the performance of the partition-based nearest neighbor data structure. Indeed, if a query is chosen as a uniformly random data point, then the average k-NN accuracy is exactly equal to the fraction of edges of the k-NN graph G whose endpoints are separated by the partition $\mathcal{P}$. This generalizes to out-of-sample queries provided that the query and dataset distributions are close, and the test accuracy of the trained classifier is high.

At the same time, our approach is directly related to and inspired by recent theoretical work [8, 9] on NNS for general metric spaces. In particular, using the framework of [8, 9], we prove that, under mild conditions on the dataset P, the k-NN graph of P can be partitioned with a hyperplane into two parts of comparable size such that only few edges get split by the hyperplane. This gives a partial theoretical justification of our method.

The new framework is very flexible and uses partitioning and learning in a black-box way. This allows us to plug various models (linear models, neural networks, etc.) and explore the trade-off between the quality and the algorithmic efficiency of the resulting partitions. We emphasize the importance of balanced partitions for

the indexing problem, where all bins contain roughly the same number of data points. This property is crucial in the distributed setting, since we naturally would like to assign a similar number of points to each machine. Furthermore, balanced partitions allow tighter control of the number of candidates simply by varying the number of retrieved parts. Note that a priori, it is unclear how to partition $\mathbb{R}^d$ so as to induce balanced bins of a given dataset. Here the combinatorial portion of our approach is particularly useful, as balanced graph partitioning is a well-studied problem, and our supervised extension to $\mathbb{R}^d$ naturally preserves the balance by virtue of attaining high training accuracy.

We speculate that the new method might be potentially useful for solving the NNS problem for non-Euclidean metrics, such as the edit distance [55] or optimal transport distance [34]. Indeed, for any metric space, one can compute the k-NN graph and then partition it. The only step that needs to be adjusted to the specific metric at hand is the learning step.

Let us finally put forward the challenge of scaling our method up to billion-sized or even larger datasets. For such scale, one needs to build an approximate k-NN graph as well as using graph partitioning algorithms that are faster than KaHIP. We leave this exciting direction to future work. For the current experiments (datasets of size 10^6 points), preprocessing takes several hours. Another important challenge is to obtain NNS algorithms based on the above partitioning with provable guarantees in terms of approximation and running time. However, we expect it to be difficult, in particular, since all the current state-of-the-art NNS algorithms lack such guarantees (e.g., k-means-based [29] or graph methods [43], see also [3] for a recent SOTA survey).

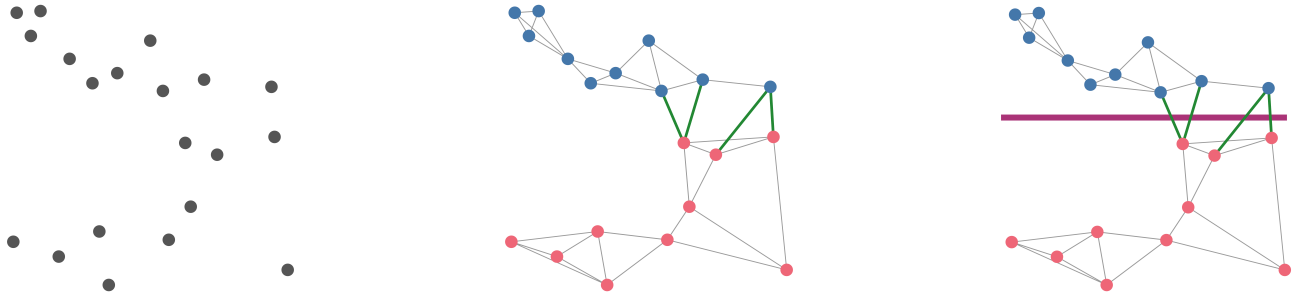
Evaluation We instantiate our framework with the KaHIP algorithm [50] for the graph partitioning step, and either linear models or small-size neural networks for the learning step. We evaluate it on several standard benchmarks for NNS [3] and conclude that in terms of quality of the resulting partitions, it consistently outperforms quantization-based and tree-based partitioning procedures, while maintaining comparable algorithmic efficiency. In the high accuracy regime, our framework yields partitions that lead to processing up to $2.3\times$ fewer candidates than the strongest baseline.

As a baseline method we use k-means clustering [29]. It produces a partition of the dataset into k bins, in a way that naturally extends to all of $\mathbb{R}^d$, by assigning a query point q to its nearest centroid. (More generally, for multi-probe querying, we can rank the bins by the distance of their centroids to q). This simple scheme yields very high-quality results for indexing. Besides k-means, we evaluate LSH [6], ITQ [23], PCA tree [49], RP tree [20], and Neural Catalyzer [48].

1.2 Related work

On the empirical side, currently the fastest indexing methods for the NNS problem are greedy graph-based algorithms, such as HNSW [43], NSG [21] or DiskANN [30]. Their high-level idea is to construct a graph on the dataset (it can be the k-NN graph, but other constructions are also possible), and then for each query perform a walk, which eventually converges to the nearest neighbor. Although very fast, graph-based approaches have suboptimal "locality of reference", which makes them less suitable for several modern architectures. For instance, this is the case when the algorithm is run on a GPU [28], or when the data is stored in external memory [SWQ+14] or in a distributed manner [12, 44]. Moreover, graph-based indexing requires many rounds of adaptive access to the dataset, whereas partition-based indexing accesses the dataset in one shot. This is crucial, for example, for nearest neighbor search over encrypted data [16]. These benefits justify further study of partition-based methods.

Machine learning techniques are particularly useful for the sketching approach, leading to a vast body of research under the label "learning to hash" [53, 54]. In particular, several recent works employed neural networks to obtain high-quality sketches [38, 48]. The fundamental difference from our work is that sketching is designed to speed up linear scans over the dataset, by reducing the cost of distance evaluation, while indexing is designed for sublinear time searches, by reducing the number of distance evaluations.



(a) Dataset

(b) k-NN graph together with a balanced partition

(c) Learned partition

Figure 1: Stages of our framework

We note that while sketches are not designed for indexing, they can be used for that purpose, since a b-bit hashing scheme induces a partition of $\mathbb{R}^d$ into 2^b parts. Nonetheless, our experiments show that partitions induced by these methods (such as Iterative Quantization [23]) are not well-suited for indexing, and underperform compared to quantization-based indexing, as well as to our methods.

We highlight in particular the recent work of [48], which uses neural networks to learn a mapping $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ that improves the geometry of the dataset and the queries to facilitate subsequent sketching. It is natural to ask whether the same family of maps can be applied to enhance the quality of partitions for indexing. However, as our experiments show, in the high accuracy regime the maps learned using the algorithm of [48] consistently degrade the quality of partitions.

Finally, we mention that here is some prior work on learning space partitions: [17, 39, 47]. However, all these algorithms learn hyperplane partitions into two parts (then applying them recursively). Our method, on the other hand, is much more flexible, since neural networks allow us to learn a much richer class of partitions.

2 Our method

Given a dataset $P \subseteq \mathbb{R}^d$ of n points, and a number of bins $m > 0$, our goal is to find a partition $\mathcal{R}$ of $\mathbb{R}^d$ into m bins with the following properties:

1. Balanced: The number of data points in each bin is not much larger than n/m.
2. Locality sensitive: For a typical query point $q \in \mathbb{R}^d$, most of its nearest neighbors belong to the same bin of $\mathcal{R}$. We assume that queries and data points come from similar distributions.
3. Simple: The partition should admit a compact description and, moreover, the point location process should be computationally efficient. For example, we might look for a space partition induced by hyperplanes.

Formally, we want the partition $\mathcal{R}$ that minimizes the loss $E_q \left[\sum_{p \in N_k(q)} \mathbf{1}_{\mathcal{R}(p) \neq \mathcal{R}(q)} \right]$ s.t. $\forall_{p \in P} \ |\mathcal{R}(p)| \leq (1+\eta)(n/m)$, where q is sampled from the query distribution, $N_k(q) \subset P$ is the set of its k nearest neighbors in P, $\eta > 0$ is a balance parameter, and $\mathcal{R}(p)$ denotes the part of $\mathcal{R}$ that contains p.

First, suppose that the query is chosen as a uniformly random data point, $q \sim P$. Let G be the k-NN graph of P, whose vertices are the data points, and each vertex is connected to its k nearest neighbors. Then the above problem boils down to partitioning vertices of the graph G into m bins such that each bin contains roughly n/m vertices, and the number of edges crossing between different bins is as small as possible (see Figure 1(b)). This balanced graph partitioning problem is extremely well-studied, and there are available combinatorial partitioning solvers that produce very high-quality solutions. In our implementation, we use the open-source solver KaHIP [50], which is based on a sophisticated local search.

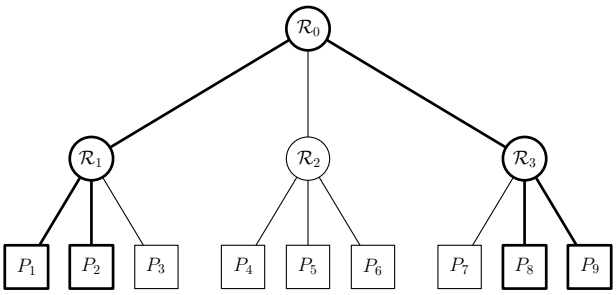


Figure 2: Hierarchical partition into 9 bins with $m_1 = m_2 = 3$. $\mathcal{R}_i$'s are partitions, P_j's are the bins of the dataset. Multi-probe query procedure, which descends into 2 bins, may visit the bins marked in bold.

More generally, we need to handle out-of-sample queries, i.e., which are not contained in P. Let $\widetilde{\mathcal{R}}$ denote the partition of G (equivalently, of the dataset P) found by the graph partitioner. To convert $\widetilde{\mathcal{R}}$ into a solution to our problem, we need to extend it to a partition $\mathcal{R}$ of the whole space $\mathbb{R}^d$ that would work well for query points. In order to accomplish this, we train a model that, given a query point $q \in \mathbb{R}^d$, predicts which of the m bins of $\widetilde{\mathcal{R}}$ the point q belongs to (see Figure 1(c)). We use the dataset P as a training set, and the partition $\widetilde{\mathcal{R}}$ as the labels – i.e., each data point is labeled with the ID of the bin of $\widetilde{\mathcal{R}}$ containing it. The method is summarized in Algorithm 1. The geometric intuition for this learning step is that – even though the partition $\widetilde{\mathcal{R}}$ is obtained by combinatorial means, and in principle might consist of ill-behaved subsets of $\mathbb{R}^d$ – in most practical scenarios, we actually expect it to be close to being induced by a simple partition of the ambient space. For example, if the dataset is fairly well-distributed on the unit sphere, and the number of bins is $m = 2$, a balanced cut of G should be close to a hyperplane.

The choice of model to train depends on the desired properties of the output partition $\mathcal{R}$. For instance, if we are interested in a hyperplane partition, we can train a linear model using SVM or regression. In this paper, we instantiate the learning step with both linear models and small-sized neural networks. Here, there is natural tension between the size of the model we train and the accuracy of the resulting classifier, and hence the quality of the partition we produce. A larger model yields better NNS accuracy, at the expense of computational efficiency. We discuss this in Section 3.

Multi-probe querying Given a query point q, the trained model can be used to assign it to a bin of a partition $\mathcal{R}$, and search for nearest neighbors within the data points in that part. In order to achieve high search accuracy, we actually train the model to predict several bins for a given query point, which are likely to contain nearest neighbors. For neural networks, this can be done naturally by taking several largest outputs of the last layer. By searching through more bins (in the order of preference predicted by the model) we can achieve better accuracy, allowing for a trade-off between computational resources and accuracy.

Hierarchical partitions When the required number of bins m is large, in order to improve the efficiency of the resulting partition, it pays off to produce it in a hierarchical manner. Namely, we first find a partition of $\mathbb{R}^d$ into m_1 bins, then recursively partition each of the bins into m_2 bins, and so on, repeating the partitioning for L levels. The total number of bins in the overall partition is $m = m_1 \cdot m_2 \cdot \ldots m_L$. See Figure 2 for illustration. The advantage of such a hierarchical partition is that it is much simpler to navigate than a one-shot partition with m bins.

Neural LSH with soft labels In the primary instantiation of our framework, we set the supervised learning component to a a neural network with a small number of layers and constrained hidden dimensions (the exact parameters are specified in the next section). In order to support effective multi-probe querying, we need to infer

Algorithm 1: Nearest neighbor search with a learned space partition

Preprocessing

Input: Dataset $P \subset \mathbb{R}^d$, integer parameter $k > 0$, number of bins $m > 0$

1: Build a k-NN graph G of P.
2: Run a balanced graph partitioning algorithm on G into m parts. Number the parts arbitrarily as $1, \ldots, m$. Let $\pi(p) \in \{1, \ldots, m\}$ denote the part containing p, for every $p \in P$.
3: Train a machine learning model M with training set P and labels $\{\pi(p)\}_{p \in P}$. For every $x \in \mathbb{R}^d$, let $M(x) \in \{1, \ldots, m\}$ denote the prediction of M on x. =0

$M(\cdot)$ defines our m-way partition of $\mathbb{R}^d$. Note that it is possible that $\pi(p) \neq M(p)$ for some $p \in P$, if M attains imperfect training accuracy.

Query

Input: query point $q \in \mathbb{R}^d$, number of bins to search b

1: Run inference on M to compute $M(q)$.
2: Search for a near neighbor of q in the bin $M(q)$, i.e., among the candidates $\{p \in P : M(p) = M(q)\}$.
3: If M furthermore predicts a distribution over bins, search for a near neighbor in the b top-ranked bins according to the ranking induced by the distribution (i.e., from the most likely bin to less likely ones). =0

not just the bin that contains the query point, but rather a distribution over bins that are likely to contain this point and its neighbors. A T-probe candidate list is then formed from all data points in the T most likely bins. In order to accomplish this, we use soft labels for data points generated as follows. For $S \geq 1$ and a data point p, the soft label $\mathcal{P} = (p_1, p_2, \ldots, p_m)$ is a distribution over the bin containing a point chosen uniformly at random among S nearest neighbors of p (including p itself). Now, for a predicted distribution $\mathcal{Q} = (q_1, q_2, \ldots, q_m)$, we seek to minimize the KL divergence between $\mathcal{P}$ and $\mathcal{Q}$: $\sum_{i=1}^{m} p_i \log \frac{p_i}{q_i}$. Intuitively, soft labels help guide the neural network with information about multiple bin ranking. S is a hyperparameter that needs to be tuned; we study its setting in the appendix (cf. Figure 6b).

3 Sparse hyperplane-induced cuts in k-NN graphs

We state and prove a theorem that shows, under certain mild assumptions, that the k-NN graph of a dataset $P \subseteq \mathbb{R}^d$ can be partitioned by a hyperplane such that the induced cut is sparse (i.e., has few crossing edges while the sizes of two parts are similar). The theorem is based on the framework of [8, 9] and uses spectral techniques.

We start with some notation. Let $N_k(p)$ be the set of k nearest neighbors of p in P. The degree of p in the k-NN graph is $\deg(p) = |N_k(p) \cup \{p' \in P \mid p \in N_k(p')\}|$. Let $\mathcal{D}$ be the distribution over the dataset P, where a point $p \in P$ is sampled with probability proportional to its degree $\deg(p)$. Let $\mathcal{D}_{\text{close}}$ be the distribution over pairs $(p, p') \in P \times P$, where $p \in P$ is uniformly random, and p' is a uniformly random element of $N_k(p)$. Denote $\alpha = \mathrm{E}_{(p,p') \in \mathcal{D}_{\text{close}}}[\|p - p'\|_2^2]$ and $\beta = \mathrm{E}_{x_1 \sim \mathcal{D}, x_2 \sim \mathcal{D}}[\|p_1 - p_2\|_2^2]$. We will proceed assuming that α (typical distance between a data point and its nearest neighbors) is noticeably smaller than β (typical distance between two independent data points).

The following theorem implies, informally speaking, that if $\alpha \ll \beta$, then there exists a hyperplane which splits the dataset into two parts of not too different size while separating only few pairs of (p, p'), where p' is one of the k nearest neighbors of p. For the proof of the theorem, see [18].

Theorem 3.1: There exists a hyperplane $H = \{x \in \mathbb{R}^d \mid \langle a, x \rangle = b\}$ such that the following holds. Let $P = P_1 \cup P_2$ be the partition of P induced by H: $P_1 = \{p \in P \mid \langle a, p \rangle \leq b\}$, $P_2 = \{p \in P \mid \langle a, p \rangle > b\}$. Then,

one has:

$$\frac{\Pr_{(p,p')\sim\mathcal{D}_{\text{close}}}[p \text{ and } p' \text{ are separated by } H]}{\min\{\Pr_{p\sim\mathcal{D}}[p \in P_1], \Pr_{p\sim\mathcal{D}}[p \in P_2]\}} \leq \sqrt{\frac{2\alpha}{\beta}}. \tag{1}$$

4 Experiments

Datasets For the experimental evaluation, we use three standard ANN benchmarks [3]: SIFT (image descriptors, 1M 128-dimensional points), GloVe (word embeddings [46], approximately 1.2M 100-dimensional points, normalized), and MNIST (images of digits, 60K 784-dimensional points). All three datasets come with 10 000 query points, which are used for evaluation. We include the results for SIFT and GloVe in the main text, and MNIST in [18].

Evaluation metrics We mainly investigate the trade-off between the number of candidates generated for a query point, and the k-NN accuracy, defined as the fraction of its k nearest neighbors that are among those candidates. The number of candidates determines the processing time of an individual query. Over the entire query set, we report both the average as well as the 0.95-th quantile of the number of candidates. The former measures the throughput[1] of the data structure, while the latter measures its latency.[2] We focus on parameter regimes that yield k-NN accuracy of at least 0.75, in the setting $k = 10$. Additional results with broader regimes of accuracy and of k are included in the appendix.

Our methods We evaluate two variants of our method, with two different choices of the supervised learning component:

- **Neural LSH:** In this variant we use small neural networks. We compare this method with k-means clustering, Iterative Quantization (ITQ) [23], Cross-polytope LSH [6], and Neural Catalyzer [48] composed over k-means clustering. We evaluate partitions into 16 bins and 256 bins. We test both one-level (non-hierarchical) and two-level (hierarchical) partitions. Queries are multi-probe.
- **Regression LSH:** This variant uses logistic regression as the supervised learning component and, as a result, produces very simple partitions induced by hyperplanes. We compare this method with PCA trees [1, 35, 49], random projection trees [20], and recursive bisections using 2-means clustering. We build trees of hierarchical bisections of depth up to 10 (thus total number of leaves up to 1024). The query procedure descends a single root-to-leaf path and returns the candidates in that leaf.

4.1 Implementation details

Neural LSH uses a fixed neural network architecture for the top-level partition, and a fixed architecture for all second-level partitions. Both architectures consist of several blocks, where each block is a fully-connected layer + batch normalization [26] + ReLU activations. The final block is followed by a fully-connected layer and a softmax layer. The resulting network predicts a distribution over the bins of the partition. The only difference between the top-level network the second-level network architecture is their number of blocks (b) and the size of their hidden layers (s). In the top-level network we use $b = 3$ and $s = 512$. In the second-level networks we use $b = 2$ and $s = 390$. To reduce overfitting, we use dropout with probability 0.1 during training. The networks are trained using the Adam optimizer [31] for under 20 epochs on both levels. We reduce the learning rate multiplicatively at regular intervals. The weights are initialized with Glorot initialization [22]. To tune soft labels, we try different values of S between 1 and 120.

[1] Number of queries per second.

[2] Maximum time per query, modulo a small fraction of outliers.

		GloVe		SIFT	
		Averages	0.95-quantiles	Averages	0.95-quantiles
One level	16 bins	1.745	2.125	1.031	1.240
	256 bins	1.491	1.752	1.047	1.348
Two levels	16 bins	2.176	2.308	1.113	1.306
	256 bins	1.241	1.154	1.182	1.192

Figure 3: Largest ratio between the number of candidates for Neural LSH and k-means over the settings where both attain the same target 10-NN accuracy, over accuracies of at least 0.85. See details in Section 4.2.

We evaluate two settings for the number of bins in each level, $m = 16$ and $m = 256$ (leading to a total number of bins of the total number of bins in the two-level experiments are $16^2 = 256$ and $256^2 = 65\,536$, respectively). In the two-level setting with $m = 256$ the bottom level of Neural LSH uses k-means instead of a neural network, to avoid overfitting when the number of points per bin is tiny. The other configurations (two-levels with $m = 16$ and one-level with either $m = 16$ or $m = 256$) we use Neural LSH at all levels.

We slightly modify the KaHIP partitioner to make it more efficient on the k-NN graphs. Namely, we introduce a hard threshold of 2000 on the number of iterations for the local search part of the algorithm, which speeds up the partitioning dramatically, while barely affecting the quality of the resulting partitions.

4.2 Comparison with multi-bin methods

Figure 4 shows the empirical comparison of Neural LSH with k-means clustering, ITQ, Cross-polytope LSH, and Neural Catalyzer composed over k-means clustering. It turns out that k-means is the strongest among these baselines.[3] The points depicted in Figure 4 are those that attain accuracy ≥ 0.75. In [18] we include the full accuracy range for all methods.

In all settings considered, Neural LSH yields consistently better partitions than k-means.[4] Depending on the setting, k-means requires significantly more candidates to achieve the same accuracy:

- Up to 117% more for the average number of candidates for GloVe;
- Up to 130% more for the 0.95-quantiles of candidates for GloVe;
- Up to 18% more for the average number of candidates for SIFT;
- Up to 34% more for the 0.95-quantiles of candidates for SIFT;

Figure 3 lists the largest multiplicative advantage in the number of candidates of Neural LSH compared to k-means, for accuracy values of at least 0.85. Specifically, for every configuration of k-means, we compute the ratio between the number of candidates in that configuration and the number of candidates of Neural LSH in its optimal configuration, among those that attained at least the same accuracy as that k-means configuration.

We also note that in all settings except two-level partitioning with $m = 256$,[5] Neural LSH produces partitions for which the 0.95-quantiles for the number of candidates are very close to the average number of candidates, which indicates very little variance between query times over different query points. In contrast, the respective gap

[3] It is important to note that ITQ is not designed to produce space partitions; as explained in Section 1, it does so as a side-effect. Simiarly, Neural Catalyzer is not designed to enhance partitions. The comparison is intended to show that they do not outperform indexing techniques despite being outside their intended application.

[4] We note that two-level partitioning with $m = 256$ is the best performing configuration of k-means, for both SIFT and GloVe, in terms of the minimum number of candidates that attains 0.9 accuracy. Thus we evaluate this baseline at its optimal performance.

[5] As mentioned earlier, in this setting Neural LSH uses k-means at the second level, due to the large overall number of bins compared to the size of the datasets. This explains why the gap between the average and the 0.95-quantile number of candidates of Neural LSH is larger for this setting.

in the partitions produced by k-means is much larger, since unlike Neural LSH, it does not directly favor balanced partitions. This implies that Neural LSH might be particularly suitable for latency-critical NNS applications.

Model sizes. The largest model size learned by Neural LSH is equivalent to storing about ≈ 5700 points for SIFT, or ≈ 7100 points for GloVe.This is considerably larger than k-means with $k \leq 256$, which stores at most 256 points. Nonetheless, we believe the larger model size is acceptable for Neural LSH, for the following reasons. First, in most of the NNS applications, especially for the distributed setting, the bottleneck in the high accuracy regime is the memory accesses needed to retrieve candidates and the further processing (such as distance computations, exact or approximate). The model size is not a hindrance as long as does not exceed certain reasonable limits (e.g., it should fit into a CPU cache). Neural LSH significantly reduces the memory access cost, while increasing the model size by an acceptable amount. Second, we have observed that the quality of the Neural LSH partitions is not too sensitive to decreasing the sizes the hidden layers. The model sizes we report are, for the sake of concreteness, the largest ones that still lead to improved performance. Larger models do not increase the accuracy, and sometimes decrease it due to overfitting.

4.3 Comparison with tree-based methods

Next we compare binary decision trees, where in each tree node a hyperplane is used to determine which of the two subtrees to descend into. We generate hyperplanes with the following methods: Regression LSH, the Learned KD-tree of [17], the Boosted Search Forest of [39], cutting the dataset into two equal halves along the top PCA direction [35, 49], 2-means clustering, and random projections of the centered dataset [20, 33]. We build trees of depth up to 10, which correspond to hierarchical partitions with the up to $2^{10} = 1024$ bins. Results for GloVe and SIFT are summarized in Figure 5 (see appendix). For random projections, we run each configuration 30 times and average the results.

For GloVe, Regression LSH significantly outperforms 2-means, while for SIFT, Regression LSH essentially matches 2-means in terms of the average number of candidates, but shows a noticeable advantage in terms of the 0.95-percentiles. In both instances, Regression LSH significantly outperforms PCA tree, and all of the above methods dramatically improve upon random projections.

Note, however, that random projections have an additional benefit: in order to boost search accuracy, one can simply repeat the sampling process several times and generate an ensemble of decision trees instead of a single tree. This allows making each individual tree relatively deep, which decreases the overall number of candidates, trading space for query time. Other considered approaches (Regression LSH, 2-means, PCA tree) are inherently deterministic, and boosting their accuracy requires more care: for instance, one can use partitioning into blocks as in [29], or alternative approaches like [33]. Since we focus on individual partitions and not ensembles, we leave this issue out of the scope.

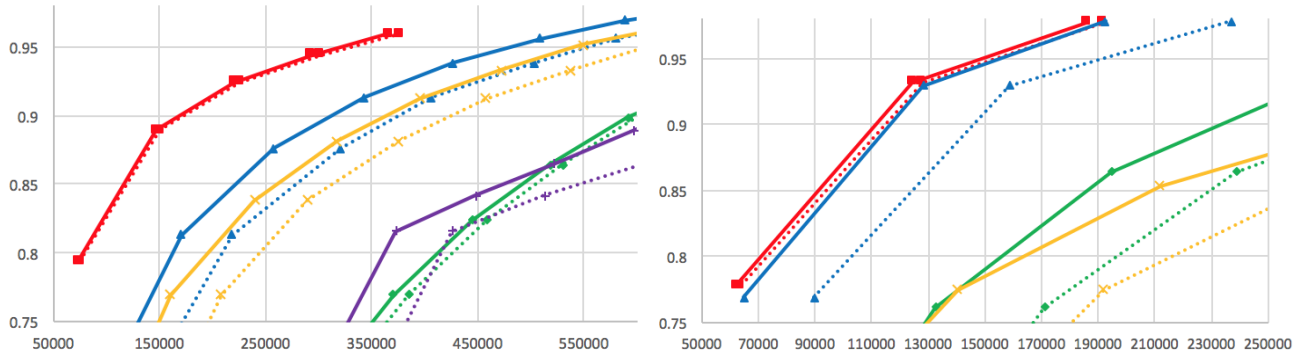
4.4 Additional experiments

In this section we include several additional experiments.

First, we study the effect of setting k. We evaluate the 50-NN accuracy of Neural LSH when the partitioning step is run on either the 10-NN or the 50-NN graph.[6] We compare both algorithms to k-means with $k = 50$. Figure 6a compares these three algorithms on GloVe for 16 bins reporting average numbers of candidates. From this plot, we can see that for $k = 50$, Neural LSH convincingly outperforms k-means, and whether we use 10-NN or 50-NN graph matters very little.

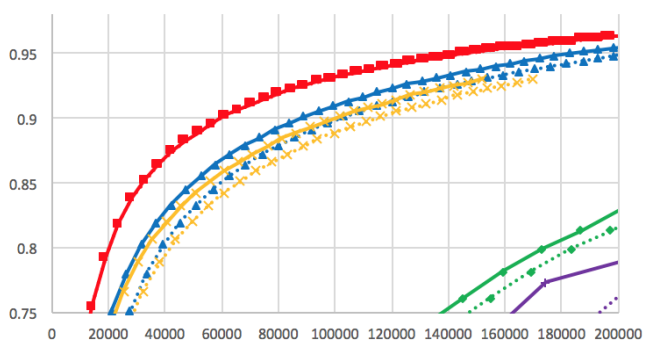
Second, we study the effect of varying S (the soft labels parameter) for Neural LSH on GloVe for 256 bins. See Figure 6b where we report the average number of candidates. As we can see from the plot, the setting $S = 15$

[6] Neural LSH can solve k-NNS by partitioning the k'-NN graph, for any k, k'; they do not have to be equal.

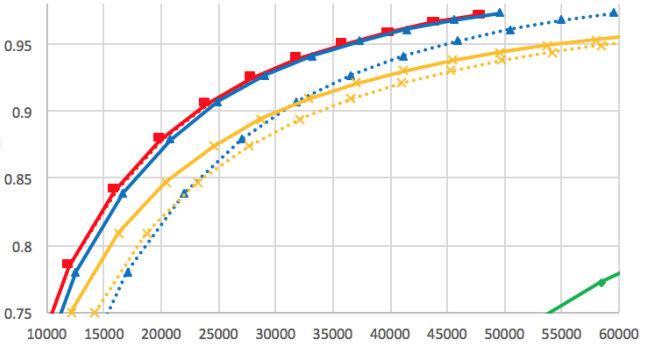


(a) GloVe, one level, 16 bins

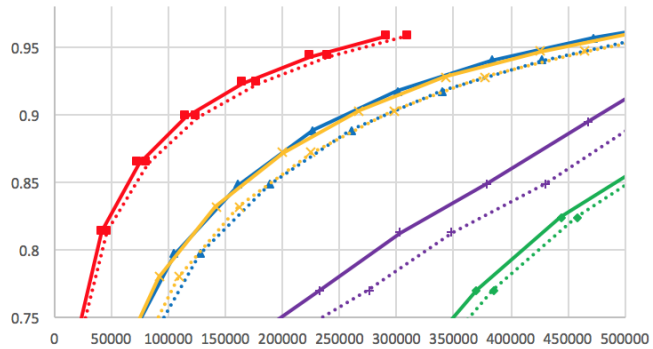
(b) SIFT, one level, 16 bins



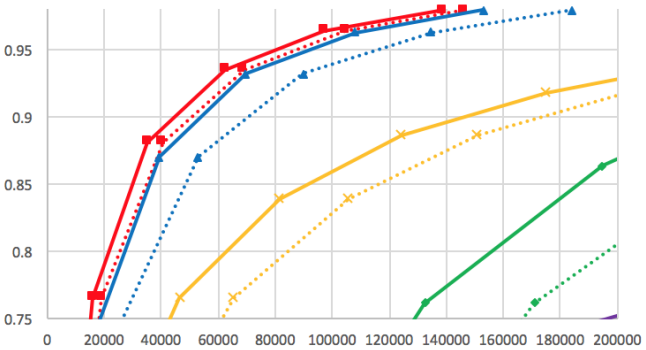
(c) GloVe, one level, 256 bins



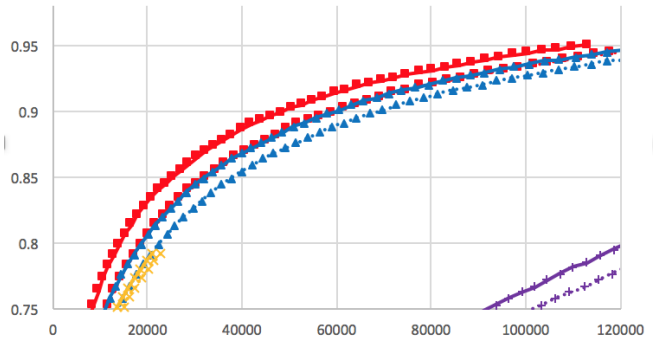
(d) SIFT, one level, 256 bins



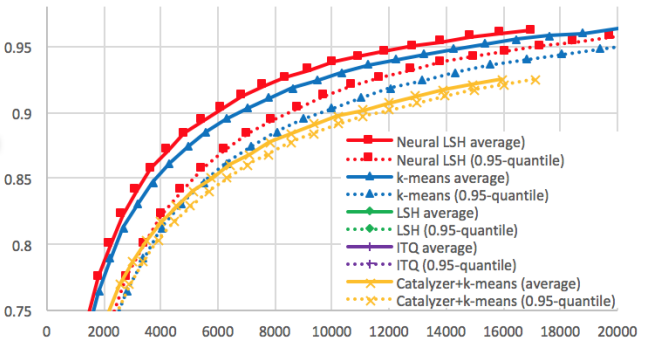
(e) GloVe, two levels, 16 bins



(f) SIFT, two levels, 16 bins



(g) GloVe, two levels, 256 bins, k-means at 2nd level



(h) SIFT, two levels, 256 bins, k-means at 2nd level

Figure 4: Comparison of Neural LSH with baselines; x-axis is the number of candidates, y-axis is the 10-NN accuracy

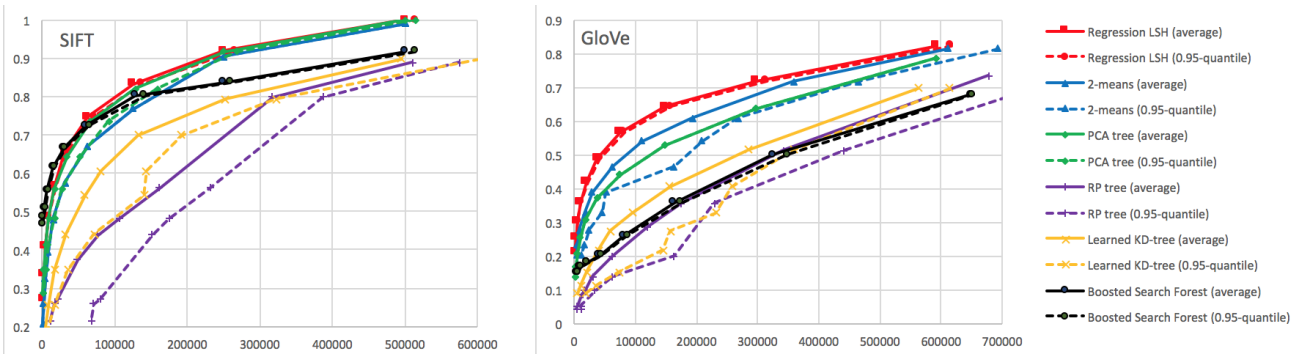
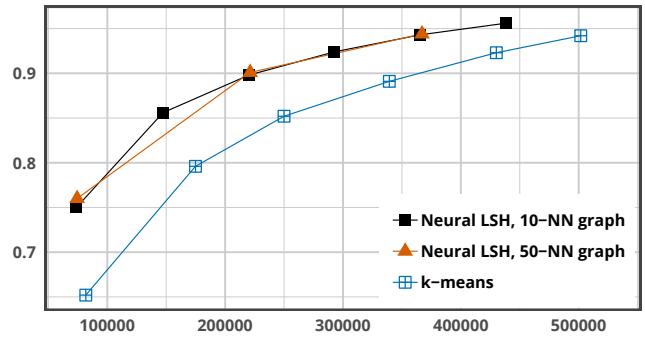
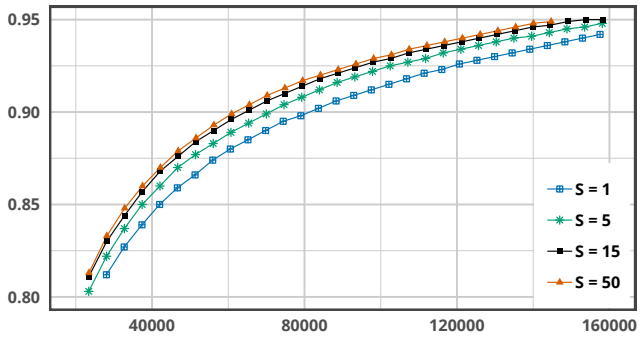


Figure 5: Comparison of decision trees built from hyperplanes: x-axis – number of candidates, y-axis – 10-NN accuracy



(a) GloVe, one level, 16 bins, 50-NN accuracy using 10-NN and 50-NN graphs



(b) GloVe, one level, 256 bins, varying S

Figure 6: Effect of various hyperparameters

yields much better results compared to the vanilla case of $S = 1$. However, increasing S beyond 15 brings diminishing returns on the overall accuracy.

5 Conclusions and future directions

We presented a new technique for finding partitions of $\mathbb{R}^d$ which support high-performance indexing for sublinear-time NNS. It proceeds in two major steps: (1) We perform a combinatorial balanced partitioning of the k-NN graph of the dataset; (2) We extend the resulting partition to the whole ambient space $\mathbb{R}^d$ by using supervised classification (such as logistic regression, neural networks, etc.). Our experiments show that the new approach consistently outperforms quantization-based and tree-based partitions.

Our work leads to multiple exciting open problems. Perhaps the most important one is whether it is possible to design a variant of Neural LSH that has provable correctness guarantees, without sacrificing the empirical performance. Such guarantees could take multiple forms:

1. approximate nearest neighbor: the data structure guarantees that, given a query q, it returns a point p' whose distance to q is at most some factor $c > 1$ greater than the distance from q to its true nearest neighbor.

2. probabilistic near neighbor: for a given scale parameter r and probability parameter $\delta > 0$, given a query q, each point p within a distance r from q is returned with probability $1 - \delta$.

Algorithms based on locality-sensitive hashing satisfy guarantees (1) [25] or (2) [4], depending on the implementation. Similarly, some tree-based methods such as [10] and [20] satisfy such guarantees. It is thus plausible that one could achieve SOTA empirical performance while guaranteeing some form of correctness. At the same time, we expect this challenge to be somewhat difficult, as the current state-of-the-art indexing algorithms typically lack correctness guarantees. For example, [27] has recently demonstrated that there exist data sets for which graph-based algorithms such as HNSW [43], NSG [21] or DiskANN [30] must scan a large percentage of data point to return a reasonable approximate nearest neighbor. Nevertheless, an initial progress towards this goal has been already made in [2].

References

[1] Amirali Abdullah, Alexandr Andoni, Ravindran Kannan, and Robert Krauthgamer, Spectral approaches to nearest neighbor search, arXiv preprint arXiv:1408.0751 (2014).

[2] Alexandr Andoni and Daniel Beaglehole, Learning to hash robustly, guaranteed, International Conference on Machine Learning, PMLR, 2022, pp. 599–618.

[3] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull, Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms, International Conference on Similarity Search and Applications, Springer, 2017, pp. 34–49.

[4] Alexandr Andoni, Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab Mirrokni, Locality-sensitive hashing using stable distributions, Nearest-Neighbor Methods in Learning and Vision (2005), 61–72.

[5] Alexandr Andoni and Piotr Indyk, Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions, Communications of the ACM **51** (2008), no. 1, 117–122.

[6] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt, Practical and optimal lsh for angular distance, Advances in Neural Information Processing Systems, 2015, pp. 1225–1233.

[7] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn, Approximate nearest neighbor search in high dimensions, arXiv preprint arXiv:1806.09823 (2018).

[8] Alexandr Andoni, Assaf Naor, Aleksandar Nikolov, Ilya Razenshteyn, and Erik Waingarten, Data-dependent hashing via nonlinear spectral gaps, Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (2018), 787–800.

[9] ———, Hölder homeomorphisms and approximate nearest neighbors, 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2018, pp. 159–169.

[10] Mayank Bawa, Tyson Condie, and Prasanna Ganesan, Lsh forest: self-tuning indexes for similarity search, Proceedings of the 14th international conference on World Wide Web, ACM, 2005, pp. 651–660.

[11] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik, Learning to branch, International Conference on Machine Learning, 2018.

[12] Bahman Bahmani, Ashish Goel, and Rajendra Shinde, Efficient distributed locality sensitive hashing, Proceedings of the 21st ACM international conference on Information and knowledge management, ACM, 2012, pp. 2174–2178.

[13] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis, Compressed sensing using generative models, International Conference on Machine Learning, 2017, pp. 537–546.

[14] Artem Babenko and Victor Lempitsky, The inverted multi-index, Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 3069–3076.

[15] Aditya Bhaskara and Maheshakya Wijewardena, Distributed clustering via lsh based data partitioning, International Conference on Machine Learning, 2018, pp. 569–578.

[16] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M Sadegh Riazi, Sanns: Scaling up secure approximate k-nearest neighbors search, arXiv preprint arXiv:1904.02033 (2019).

[17] Lawrence Cayton and Sanjoy Dasgupta, A learning framework for nearest neighbor search, Advances in Neural Information Processing Systems, 2007, pp. 233–240.

[18] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner, Learning space partitions for nearest neighbor search, arXiv preprint arXiv:1901.08544 (2019).

[19] Hanjun Dai, Elias Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song, Learning combinatorial optimization algorithms over graphs, Advances in Neural Information Processing Systems, 2017, pp. 6351–6361.

[20] Sanjoy Dasgupta and Kaushik Sinha, Randomized partition trees for exact nearest neighbor search, Conference on Learning Theory, 2013, pp. 317–337.

[21] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai, Fast approximate nearest neighbor search with the navigating spreading-out graph, Proceedings of the VLDB Endowment **12** (2019), no. 5, 461–474.

[22] Xavier Glorot and Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks, International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

[23] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin, Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval, IEEE Transactions on Pattern Analysis and Machine Intelligence **35** (2013), no. 12, 2916–2929.

[GSL+20] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar, Accelerating large-scale inference with anisotropic vector quantization, International Conference on Machine Learning, 2020.

[25] Piotr Indyk and Rajeev Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998, pp. 604–613.

[26] Sergey Ioffe and Christian Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167 (2015).

[27] Piotr Indyk and Haike Xu, Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations, manuscript (2023).

[28] Jeff Johnson, Matthijs Douze, and Hervé Jégou, Billion-scale similarity search with gpus, arXiv preprint arXiv:1702.08734 (2017).

[29] Herve Jégou, Matthijs Douze, and Cordelia Schmid, Product quantization for nearest neighbor search, IEEE transactions on pattern analysis and machine intelligence **33** (2011), no. 1, 117–128.

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

[31] Diederik Kingma and Jimmy Ba, Adam: A method for stochastic optimization, International Conference for Learning Representations, 2015.

[32] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis, The case for learned index structures, Proceedings of the 2018 International Conference on Management of Data, ACM, 2018, pp. 489–504.

[33] Omid Keivani and Kaushik Sinha, Improved nearest neighbor search using auxiliary information and priority functions, International Conference on Machine Learning, 2018, pp. 2578–2586.

[34] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger, From word embeddings to document distances, International Conference on Machine Learning, 2015, pp. 957–966.

[35] Neeraj Kumar, Li Zhang, and Shree Nayar, What is a good nearest neighbors algorithm for finding similar patches in images?, European conference on computer vision, Springer, 2008, pp. 364–378.

[36] Jinfeng Li, James Cheng, Fan Yang, Yuzhen Huang, Yunjian Zhao, Xiao Yan, and Ruihao Zhao, Losha: A general framework for scalable locality sensitive hashing, Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2017, pp. 635–644.

[37] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li, Multi-probe lsh: efficient indexing for high-dimensional similarity search, Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment, 2007, pp. 950–961.

[38] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou, Deep hashing for compact binary codes learning, Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 2475–2483.

[39] Zhen Li, Huazhong Ning, Liangliang Cao, Tong Zhang, Yihong Gong, and Thomas S Huang, Learning to search efficiently in high dimensions, Advances in Neural Information Processing Systems, 2011, pp. 1710–1718.

[40] Thodoris Lykouris and Sergei Vassilvitskii, Competitive caching with machine learned advice, International Conference on Machine Learning, 2018.

[41] Michael Mitzenmacher, A model for learned bloom filters and optimizing by sandwiching, Advances in Neural Information Processing Systems, 2018. 2–1783.

[42] Ali Mousavi, Ankit B Patel, and Richard G Baraniuk, A deep learning approach to structured signal recovery, Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on, IEEE, 2015, pp. 1336–1343.

[43] Yury A Malkov and Dmitry A Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical

navigable small world graphs, IEEE transactions on pattern analysis and machine intelligence (2018).

[44] Y Ni, K Chu, and J Bradley, Detecting abuse at scale: Locality sensitive hashing at uber engineering, 2017.

[45] Manish Purohit, Zoya Svitkina, and Ravi Kumar, Improving online algorithms via ml predictions, Advances in Neural Information Processing Systems, 2018, pp. 9661–9670.

[46] Jeffrey Pennington, Richard Socher, and Christopher Manning, Glove: Global vectors for word representation, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1532–1543.

[47] Parikshit Ram and Alexander Gray, Which space partitioning tree to use for search?, Advances in Neural Information Processing Systems, 2013, pp. 656–664.

[48] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Herve Jégou, Spreading vectors for similarity search, International Conference on Learning Representations, 2019.

[49] Robert F Sproull, Refinements to nearest-neighbor searching ink-dimensional trees, Algorithmica **6** (1991), no. 1-6, 579–589.

[50] Peter Sanders and Christian Schulz, Think Locally, Act Globally: Highly Balanced Graph Partitioning, Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13), LNCS, vol. 7933, Springer, 2013, pp. 164–175.

[SWQ+14] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin, Srs: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index, Proceedings of the VLDB Endowment **8** (2014), no. 1, 1–12.

[WGS+17] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N Holtmann-Rice, David Simcha, and Felix Yu, Multiscale quantization for fast similarity search, Advances in Neural Information Processing Systems, 2017, pp. 5745–5755.

[53] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang, Learning to hash for indexing big data - a survey, Proceedings of the IEEE **104** (2016), no. 1, 34–57.

[54] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji, Hashing for similarity search: A survey, arXiv preprint arXiv:1408.2927 (2014).

[55] Haoyu Zhang and Qin Zhang, Embedjoin: Efficient edit similarity joins via embeddings, Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 585–594.

Querying Time-Series Data: A Comprehensive Comparison of Distance Measures

John Paparrizos*, Chunwei Liu‡, Aaron J. Elmore†, Michael J. Franklin†
*The Ohio State University, paparrizos.1@osu.edu
†University of Chicago
‡Massachusetts Institute of Technology

Abstract

Distance measures are core building blocks in time-series analysis and the subject of active research for decades. Unfortunately, the most detailed experimental study in this area is outdated (over a decade old) and, naturally, does not reflect recent progress. Importantly, this study (i) omitted multiple distance measures, including a classic measure in the time-series literature; (ii) considered only a single time-series normalization method; and (iii) reported only raw classification error rates without statistically validating the findings, resulting in or fueling four misconceptions in the time-series literature. Motivated by the aforementioned drawbacks and our curiosity to shed some light on these misconceptions, we comprehensively evaluate 71 time-series distance measures. Specifically, our study includes (i) 8 normalization methods; (ii) 52 lock-step measures; (iii) 4 sliding measures; (iv) 7 elastic measures; (v) 4 kernel functions; and (vi) 4 embedding measures. We extensively evaluate these measures across 128 time-series datasets using rigorous statistical analysis. For the most promising measures, we present an accuracy-to-runtime analysis and summarize recent progress on a generalized lower bounding measure that accelerates all elastic distances. Our findings debunk four long-standing misconceptions that significantly alter the landscape of what is known about existing distance measures. With the new foundations in place, we discuss open challenges and promising directions.

1 Introduction

The understanding of a multitude of natural or human-made processes involves the analysis of high-dimensional observations over time. The recording of such time-varying measurements leads in an ordered sequence of data points called time series [64, 65]. In the last decades, time-series analysis has become increasingly prevalent, affecting virtually all scientific disciplines and their corresponding industries [25, 39, 47, 56, 60, 71, 72]. With sensors and devices becoming increasingly networked and with the explosion of Internet-of-Things (IoT) applications, the volume of produced time series is expected to continue to rise [41, 42, 55, 57, 74]. This growth and ubiquity of time series generates tremendous interest in the extraction of meaningful knowledge from time series [31, 67].

The basis for most analytics over time series involves the detection of similarities between time series. The measurement of similarity, through a distance or similarity measure, is the most fundamental building block in time-series data mining, fueling tasks such as querying [2, 29, 76, 78], indexing [32, 45, 112], clustering [7, 43, 68–70], classification [6, 67, 85, 107], motif discovery [53, 62, 108], and anomaly detection [9–12, 24, 75, 77, 96]. In contrast to other data types where distance measures often process observations independently, for time series, distance measures consider sequences of observations together. This characteristic complicates the definition of distance measures for time series and, therefore, it is desirable to study the factors that determine their effectiveness.

The difficulty in formalizing accurate distance measures stems from the inability to express precisely the notion of similarity. As humans we easily recognize perceptually similar time series, by ignoring a variety

of distortions, such as fluctuations, misalignments, and stretching of observations. However, it is challenging to derive definitions to reflect the similarity for mathematically non-identical time series [33]. Due to that difficulty and the need to handle the variety of distortions, dozens of distance measures have been proposed [5, 8, 16–18, 29, 34, 35, 61, 67, 68, 88, 95, 99].

Despite this abundance of time-series distance measures and their implications in the effectiveness for a multitude of time-series tasks, less attention has been given in their comprehensive experimental validation. Specifically, in the past two decades, only a single comprehensive experimental evaluation has been dedicated to studying the accuracy of 9 influential time-series distance measures over 38 datasets [29]. Unfortunately, this study suffers from three main drawbacks: (i) this study omitted multiple distance measures, including one of the most classic measures in the time-series literature, namely, the cross-correlation measure [13, 79]; (ii) this study considered only a single time-series normalization method; and (iii) this study reported raw classification error rates without performing any rigorous statistical analysis to assess the significance of the findings. Therefore, the analysis is incomplete, and, the findings might not be conclusive. Importantly, this study is now outdated (more than a decade old), and, naturally, it does not reflect recent progress. Considering the previous drawbacks as well as the remarkable interest in time-series analysis, we believe it is critical to revisit this subject.

However, our effort is not only motivated by the necessity to address the aforementioned issues or to extend the previous study with newer datasets and distance measures. Instead, the thorough experimental evaluation of time-series distance measures that we present in this paper is the byproduct of our attempt to challenge four long-standing misconceptions (see $\mathcal{M}1 - \mathcal{M}4$ in Section 2) that have appeared in the time-series literature. These misconceptions are concerned with the (i) normalization of time series; (ii) identification of the state-of-the-art distance measure in every category of measures; (iii) performance of the omitted measures against state-of-the-art measures; and (iv) detection of the most powerful category of measures. Such misconceptions originated from several influential papers [2, 8, 34, 40, 94], some of which date back a quarter of a century, and are fueled by recent inconclusive findings [29] as well as successive claims in the literature that we discuss later. Considering how widely cited and impactful these papers are, we believe it is risky not to challenge such persistent misconceptions that might disorientate newcomer researchers and practitioners.

Motivated by the aforementioned issues and our curiosity to shed some light on these misconceptions, we conduct a comprehensive experimental evaluation to validate the effectiveness of 71 time-series distance measures. These distance measures belong to five categories: (i) 52 lock-step measures, which compare the ith point of one time series with the ith point of another; (ii) 4 sliding measures, which are the sliding versions of lock-step measures when comparing one time series with all shifted versions of the other; (iii) 7 elastic measures, which create a non-linear mapping between time series by comparing one-to-many points in order to align or stretch points; (iv) 4 kernel measures, which use a function (with lock-step, sliding, or elastic properties) to implicitly map data into a high-dimensional space; and (v) 4 embedding measures, which exploit distance or kernel measures indirectly for constructing new representations for time series. In addition, we consider 8 normalization methods for time series.

We perform an extensive evaluation of these distance measures across 128 datasets [25] and compare their classification accuracy obtained from one-nearest-neighbor classifiers (1-NN) under both supervised and unsupervised settings. We conduct a rigorous statistical validation of our findings by employing two statistical tests to assess the significance of the differences in classification accuracy when comparing pairs of measures or multiple measures together. In summary, our study identifies (i) normalization methods leading to significant improvements in a number of distance measures; (ii) new lock-step measures that significantly outperform the current state of the art; (iii) an omitted baseline that most highly popular elastic measures do not outperform; and (iv) new elastic and new kernel measures that significantly outperform the current state of the art. These findings debunk the four long-standing misconceptions and alter the landscape of what is known about existing measures.

We start with the description of the four misconceptions in the literature (Section 2) and we review the relevant background (Section 3). Then, we present our contributions:

- We explore for the first time 8 normalization methods along 56 distance measures (Section 4).

- We study 52 lock-step distance measures (Section 5).
- We investigate 4 classic sliding measures omitted from every previous evaluation (Section 6).
- We compare 7 elastic measures under supervised and unsupervised settings (Section 7).
- We study for the first time 4 kernel (Section 8) and 4 embedding distance measures (Section 9).
- We present an accuracy-to-runtime analysis (Section 10).
- We summarize recent progress towards accelerating the strongest elastic distances via the use of lower bounding measures (Section 11).

Finally, we discuss new directions (Section 12) and conclude with the implications of our work (Section 13). An earlier version of the paper has been published in ACM SIGMOD 2020 [73].

2 The Four Misconceptions

In this section, we describe four misconceptions that have appeared in the time-series literature.

These misconceptions have originated in part from several influential papers [2, 8, 34, 40, 94]. Subsequently, these misconceptions were fueled by a comprehensive study of time-series distance measures [29] as well as dozens of subsequent papers in the literature trusting its findings. Even though an extension of this study appeared five years later [102], this newer version focused on elaborating on the previous results. Recent studies that have focused on time-series classification [6, 54] performed a statistical analysis of several classifiers, including the distance measures in [29, 102]. Unfortunately, these studies only considered supervised tuning of necessary parameters, which does not reflect the use of distance measures for similarity search [32]. Importantly, some results in [6] contradict other results in [54], which, in turn, validated claims that there is no significant difference between the evaluated elastic measures [29, 102]. Interestingly, the improved accuracy found for some measures was attributed to the evaluation framework used while otherwise it was claimed to be undetectable [6]. Considering such apparent difficulties in providing conclusive evidence for this important subject, it is not surprising that the following misconceptions have persisted for so long.

Before we dive into the details, we emphasize that we do not believe or imply that any of these misconceptions were created on purpose. On the contrary, we believe that they are based on evidence, trends, and resources available at the given point in time. We describe the four misconceptions in the form of answers to questions a newcomer researcher would likely identify by studying the literature.

$\mathcal{M}1$**: How to normalize time series?** The consensus is to use the z-score or z-normalization method. Starting with the work of Goldin and Kanellakis [40], a follow-up of the two seminal papers for sequence [2] and subsequence [34] search in time-series databases, that suggested first to normalize the time series to address issues with scaling and translation, z-normalization became the prevalent method to preprocess time series. Despite the proposal of alternative methods the same year [3], the z-normalization was subsequently preferred as the suggested transformations are also applicable to the widely popular Fourier representation [2, 34, 83]. Due to the ubiquity of z-normalization, a valuable resource for time series, the UCR Archive [25], offered until recently the datasets in their z-normalized form. To the best of our knowledge, no study has ever extensively evaluated normalization methods for time series. We review 8 approaches in Section 4 and study their performance in Sections 5 and 6.

$\mathcal{M}2$**: Which lock-step measure to use?** The consensus is to use the Euclidean distance (ED). ED was the method of choice in the first paper for sequence search in time series [2] due to its usefulness in many cases and its applicability over feature vectors. Considering that ED is straightforward to implement, parameter-free, efficient, as well as tightly connected with the Fourier representation and widely supported by indexing mechanisms (in contrast to other L_p-norm variants [109]), there is no surprise about its popularity. Besides, evidence that with increased dataset sizes, the classification error of ED converges to the error of more accurate measures [94], justified its use from virtually all current time-series indexing methods [32]. (Our results in Section 10 suggest that classification error of ED may not always converge to the error of more accurate measures, at least not always

with the same speed of convergence.) In Section 5, we evaluate 52 lock-step measures.

$\mathcal{M}3$: Are elastic better than sliding measures? The answer is currently unknown. Despite the wide popularity of the cross-correlation measure, also known as sliding Euclidean or dot product distance, in the signal and image processing literature [14], cross-correlation has largely been omitted from distance measure evaluations. We believe two factors are responsible for that. First, cross-correlation was considered in the seminal paper [2] as a typical similarity measure, but ED was preferred instead because (i) cross-correlation reduces to ED; and (ii) for the aforementioned reasons in $\mathcal{M}2$. Second, in the introduction of Dynamic Time Warping (DTW) [8], an elastic measure, as an alternative to ED a year later, no comparison was performed against cross-correlation, an obvious baseline. Subsequently, virtually all research on that subject focused either on lock-step or elastic measures [32, 33], with a few exceptions [52, 68, 89]. Interestingly, cross-correlation was not considered as a baseline method in any of the proposed elastic measures [16–18, 61, 95, 99], neither in any of the experimental evaluations of distance measures discussed previously [6, 29, 54, 102]. Strangely, cross-correlation was also omitted from many popular surveys [33, 84]. Therefore, it remains unknown if elastic measures outperform sliding measures. We study 4 sliding measures in Section 6 and analyse their performance against elastic measures in Section 7.

$\mathcal{M}4$: Is DTW the best elastic measure? The general consensus that has emerged is yes. Since the introduction of DTW as a distance measure for time series [8], DTW has inspired the exploration of edit-based distances and it is widely used as the baseline method for this problem [6, 16–18, 54, 61, 68, 95, 99]. It is not uncommon to identify statements even in the abstracts of papers that 1-NN with DTW is exceptionally difficult to outperform [80–82, 105]. Such statements have been backed over the years by the aforementioned extensive evaluations, which conclude that (i) the accuracy of other elastic measures is very close to that of DTW [29, 102]; (ii) there is no significant difference in the accuracy of elastic measures [54]; and (iii) that it is "a little embarrassing" that most classifiers do not outperform 1-NN with DTW [6]. Therefore, there is little space to doubt that DTW is the best elastic measure. To study that misconception, we validate 7 elastic measures in Section 7.

To complete the analysis and capture recent progress, we also include kernel measures and embedding measures in our evaluation (Sections 8 and 9). With the detailed presentation of the four misconceptions, we believe we have now convinced the reader that these misconceptions are not based on any personal biases but, instead, have originated naturally along with the evolution of this area. However, it is risky to not challenge their validity, which may result in confusion for newcomer researchers and practitioners and discourage them from tackling problems in that area. Importantly, it is surprising to consider that half a century of scientific progress has not resulted in any significant improvements over ED or the 50-year-old DTW [87].

Next, we review the relevant background required to validate the accuracy of the normalization methods and distance measures. Even though the efficiency of measures is another important factor of their effectiveness, there are many ways to accelerate each measure, ranging from hardware-aware implementations to algorithmic solutions such as the use of indexing or comparison pruning. We refer the reader to an excellent recent study of data-series similarity search [32], which shows the level of detail required to only evaluate ED. Therefore, we leave such detailed study for future work but we present an accuracy-to-runtime analysis in Section 10.

3 Preliminaries and background

In this section, we review the necessary background for our experimental evaluation.

Terminology and definitions: We consider a time-series dataset as a set of n real-valued vectors $X = [\vec{x}_1, \ldots, \vec{x}_n]^\top \in \mathbb{R}^{n \times m}$, where each time series, $\vec{x}_i \in \mathbb{R}^m$, is an m-dimensional ordered sequence of data points. From this definition, it becomes clear that we consider univariate time series of equal length, where each of these points is a scalar. Following the previous evaluations [6, 29, 102], we consider that the sampling rates of all time series are the same and omit the discrete time stamps.

Datasets: To conduct our extensive evaluation, we use one of the most valuable public resources in the time-series data mining literature, the UCR Time-Series Archive [25]. This archive contains the largest collection

of class-labeled time-series datasets. Currently, the archive consists of 128 datasets and includes time series from sensor readings, image outlines, motion capture, spectrographs, medical signals, electric devices, as well as simulated time series. Each dataset contains from 40 to $24,000$ time series, the lengths vary from 15 to $2,844$, and each time series is annotated with a single label. The majority of the datasets are already z-normalized and we apply the same normalization to all datasets.

The latest version of the archive has deliberately left a small number of datasets containing time series with varying lengths and missing values to reflect the real world. Following the recommendation of the authors of the archive, who performed similar steps to report classification accuracy numbers on the UCR archive website [25], we resample shorter time series to reach the longest time series in each dataset and we fill missing values using linear interpolation. Through these steps, we make the new datasets compatible with previous versions of the archive [66].

Evaluation framework: Following the previous studies [6, 29], we also employ the 1-NN classifier in our evaluation framework, with important differences. 1-NN classifiers are suitable methods for distance measure evaluation for several reasons [29]. Specifically, 1-NN classifiers: (i) resemble the problem solved in time-series similarity search [32]; (ii) are parameter-free and easy to implement; (iii) dependent on the choice of distance measure; and (iv) provide an easy-to-interpret (classification) accuracy measure, which captures if the query and the nearest neighbor belong to the same class.

A critical step for the effectiveness of classifiers is the splitting of a dataset into training and test sets. Previous studies [6, 29, 102] used the k-cross-validation resampling procedure, which produces k groups of time series, tunes necessary parameters on the $k-1$ groups, and evaluates the distance measures using the group of time series left. Strangely, [29, 102] tuned parameters only on a single group and evaluated the distance measures using the $k-1$ groups, which contradicts the common practice. In [6], the improved accuracy of some measures is attributed to such a resampling procedure, while otherwise, it was claimed to be undetectable. Therefore, to eliminate biases from resampling, we respect the split of training and test sets provided by the UCR archive as well as the class distribution in the datasets (i.e., some datasets contain the same number of time series in each class while other datasets contain imbalanced classes). This decision makes our evaluation framework deterministic and enables reproducibility. Refer to [73] for further details on our evaluation settings.

Statistical analysis: To assess the significance of the differences in accuracy, we employ two statistical tests to validate the pairwise comparisons of measures and the comparisons of multiple measures together. Specifically, following the highly influential [26], we use the Wilcoxon test [103] with a 95% confidence level to evaluate pairs of measures over multiple datasets, which is more appropriate than the t-test [86]. As with pairwise tests we cannot reason about multiple measures together and following [26], we also use the Friedman test [36] followed by the post-hoc Nemenyi test [63] to compare multiple measures over multiple datasets and report statistical significant results with 90% confidence level (because these tests require more evidence than Wilcoxon).

Availability of code and results: We implemented the evaluation framework in Matlab, with imported C and Java codes for several distance measures. To ensure the reproducibility of our findings, we make the code available.[1]

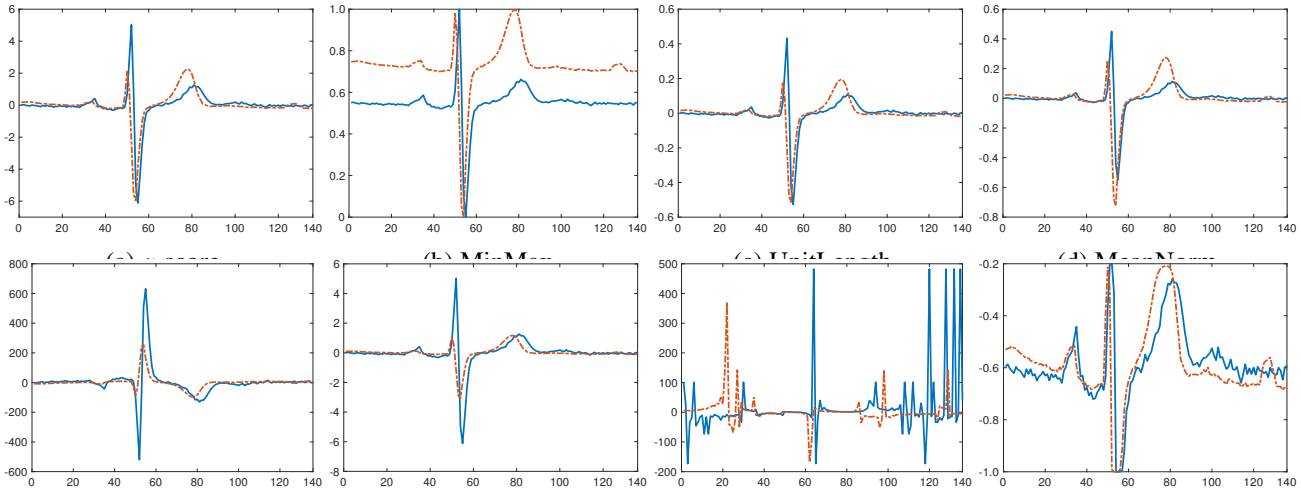
Environment: We ran our experiments on 15 identical servers: Dual Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz and 196GB RAM. Each server has 24 physical cores (12 per CPU), which provided us with 360 cores for four months.

Next, we start with the study of normalization methods.

4 Time-Series Normalizations

In this section, we review 8 normalization methods. As we discussed earlier, a critical issue when comparing time series is how to handle a number of distortions that are characteristic of the time series. For complex distortions,

[1] `https://github.com/TheDatumOrg/TSDistEval`



(e) MedianNorm (f) AdaptiveScaling (g) Logistic (h) Tanh

Figure 1: Example of how each of the 8 normalization methods transforms time series of ECGFiveDays [25].

sophisticated distance measures are required as offering invariances to such distortions is not trivial, which explains the proliferation of distance measures in the literature. However, in several cases, a simple preprocessing step is generally sufficient, as we see next.

Consider the following two examples [40]: (i) two products with similar sales patterns but different sales volume; and (ii) temperatures of two days starting at different values but exhibiting the exact same pattern. The first is an example of the difference in scale between two time series, whereas the second is an example of the difference in translation. Despite such differences, in many cases, it is useful to recognize the similarity between time series. Formally, for any constants a (scale) and b (translation), linear transformations in time series of the form $a\vec{x} + b$ should not affect their similarity.

Several methods have been proposed to handle these popular distortions. Normalization methods transform the data to become normally distributed, whereas standardization methods place different data ranges on a common scale. In the machine-learning literature, feature scaling is also used to refer to such methods. In practice, all terms are used interchangeably to refer to some data transformation.

We consider 8 popular normalization methods in our study, namely, z-score, min-max (MinMax), Mean (MeanNorm), Median (MedianNorm), Unit length (UnitLength), Adaptive scaling (AdaptiveScaling), Logistic or Sigmoid (Logistic), and Hyperbolic tangent (Tanh) normalization. (Please refer to [73] for more details and their mathematical formulas.) Figure 1, shows an example of how each one of the previously described normalization methods transforms a pair of time series from ECGFiveDays [25]. We observe that in some cases, the differences are only visible in the range of values (e.g., z-score vs. UnitLength), but, in others, the visual effect is more distinct (e.g., MinMax, MeanNorm, and AdaptiveScaling). The most unexpected visual effects come from the two non-linear transformations (i.e., Logistic and Tanh). Next, we evaluate 8 methods along with the 52 lock-step measures.

5 Time-Series Lock-Step Distances

In this section, we study 52 lock-step measures that have been proposed across different disciplines.

Distance measures provide a numerical value to quantify how distant are pairs of objects represented as points, vectors, or matrixes. Due to the difficulty in formalizing the notion of similarity, as well as the need to handle a variety of distortions and applications, hundreds of distance measures have been proposed in the literature. This proliferation of distance measures across different scientific areas has resulted in multi-year efforts to organize

this knowledge into dictionaries [27] and encyclopedias [28].

As it is understandable, not all of these measures are applicable to time-series data. Thankfully, different endeavors have already been conducted to identify appropriate measures for a variety of tasks across different fields [37, 110]. An influential study [15] identified 50 lock-step distance measures that we adapt in our evaluation of time-series distance measures. We note that a previous study [38] evaluated a subset of these measures (45) using 1-NN over 42 datasets from the UCR archive and concluded that there is no significant differences between these lock-step distance measures.

Unfortunately, we identified issues with this study. First, several of the evaluated measures are known to be equivalent to each other and, therefore, they should provide identical classification accuracy results. For example, this is the case for the Euclidean distance and the inner product (or Pearson's correlation), which under z-normalization, they should provide the same accuracy numbers. Second, several distance measures were not properly implemented, resulting in using as distance values either the real part of complex numbers or the first value of a normalized vector of the input time series. Therefore, the analysis of these lock-step measures is incomplete, and the findings are inconclusive.

In our study, we have carefully re-implemented all 50 distance measures from [15]. The distance measures belong to 7 different families of measures: (1) 4 measures belong to the L_p Minkowski family; (2) 6 measures belong to the L_1 family; (3) 7 measures belong to the Intersection family; (4) 6 measures belong to the Inner Product family; (5) 5 measures belong to the Fidelity family; (6) 8 measures belong to the L_2 family; and (7) 6 measures belong to the Entropy family. Apart from these 42 measures, we also consider the 3 measures that utilize ideas from multiple other measures (Combinations) as well as 5 measures proposed in the survey but not reported in the literature (until that point).

Besides these measures, we also include two measures that have substantial differences from the previous lock-step measures. Specifically, DISSIM [35] defines the distance as a definite integral of the function of time of the ED in order to take into consideration different sampling rates of time series. This computationally expensive operation can be approximated by a modified version of ED that considers in the distance of the ith points the $i+1$th points, which is a form of a smoothing operation. Finally, the adaptive scaling distance (ASD), embeds internally the AdaptiveScaling normalization with an inner product measure to compare time series under optimal scaling [19, 106].

Evaluation of lock-step measures: For all mathematical formulas, we refer the reader to the previous survey [15]. We evaluate 52 distance measures and their combinations with 8 normalization methods using our 1-NN classifier over 128 datasets (see Section 3). From all combinations of distance measures and normalization methods ($52 \cdot 8 = 416$ in total), we observe 14 measures with some improvement in their average accuracy in contrast to ED and overall 36 combinations with different normalization methods. However, only about half of these combinations result in statistically significant differences according to the pairwise Wilcoxon test. (Refer to [73] for raw numbers in Table 1.) To better understand the performance of lock-step measures, we also evaluate the significance of their differences in accuracy when considering several distance measures together, using the Friedman test followed by a post-hoc Nemenyi test. Specifically, we perform two analyses: (i) we evaluate different distance measures under the same normalization; and (ii) we evaluate standalone distance measures under different normalizations; Figure 2 shows the average rank across all datasets of the distance measures, which under z-score normalization, outperformed previously ED. The thick line connects measures that do not perform statistically significantly better. We observe that Lorentzian is ranked first (once we ignore the supervised Minkowski), meaning that it performed best in the majority of the datasets. All 5 measures significantly outperform ED, but we observe no difference between them. Figure 3 evaluates a standalone distance measure, the Lorentzian measure that performed the best previously, with different normalization methods against ED with z-score. We observe that the 3 out of the 4 combinations that were better than ED under the Wilcoxon test remain better under this statistical analysis, and there is no difference between them.

Debunking $\mathcal{M}1$ and $\mathcal{M}2$: Our evaluation shows clear evidence that normalization methods other than z-score can lead to significant improvements, which debunks $\mathcal{M}1$. Even though for standalone measures, we did not

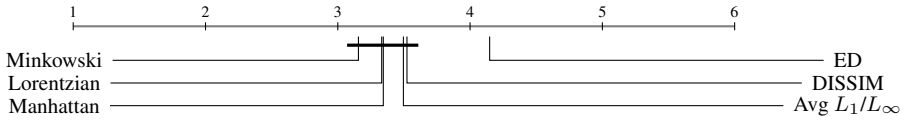


Figure 2: Ranking of lock-step measures under z-score based on the average of their ranks across datasets.

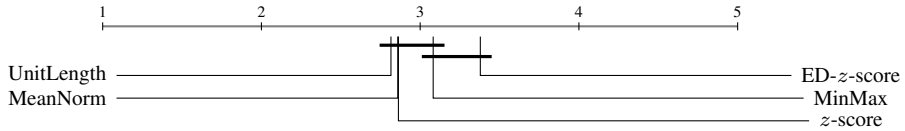


Figure 3: Ranking of normalization methods in combination with the Lorentzian distance based on the average of their ranks across datasets. ED uses z-score normalization.

observe significant improvements (e.g., ED with MeanNorm vs. ED with z-score), that does not reject our hypothesis. We note that the majority of the UCR datasets are in their z-normalized form and, therefore, for fairness, we z-normalized all datasets, which may have limited this analysis. Despite that, we identified two new distance measures, unknown until now, that only under MinMax and MeanNorm methods outperform ED with z-score and, importantly, z-score is not suitable for them. Normalizations such as MeanNorm, which combines z-score and MinMax methods, seems to perform the best for several measures. Similarly, our analysis shows that distance measures other than ED can lead to significant improvements, which debunks $\mathcal{M}2$. We identified 7 distance measures that significantly outperform ED. We emphasize that no previous study considered different normalization methods in order to challenge $\mathcal{M}1$, and our findings contradict both previous studies [29, 38], which concluded that there is no significant difference in the accuracy of lock-step measures.

Next, we focus on sliding versions of lock-step measures.

6 Time-Series Sliding Distances

We study 4 variants of cross-correlation, a measure that has largely been omitted from evaluations.

Starting with the concurrent introduction of lock-step and elastic measures for the problem of time-series similarity search [2, 8, 34], the vast majority of research focused on these two categories of measures (see $\mathcal{M}3$ in Section 2). Cross-correlation, which is similar to convolution, dates back in the 1700s [30] but received practical popularity only after the invention of Fast Fourier Transform (FFT) [20], which dramatically reduced its computational cost. Cross-correlation is one of the most fundamental operations in signal processing [14] and, lately, in deep neural networks [48, 49]. Recently, research focusing on time-series clustering used cross-correlation and achieved state-of-the-art performance for this task [68, 69]. However, this work assumed z-normalized time series and performed evaluations only against ED and DTW. (Refer to [68, 73] for the mathematical notation.)

Evaluation of sliding measures: Due to the resemblance of cross-correlation to the sliding version of Pearson's correlation, when time series are z-normalized, the majority of the literature assumes this underlying data normalization [68]. To the best of our knowledge, the performance of cross-correlation as a measure to compare time series under different normalization methods is not well explored. We measure the performance of the combinations of cross-correlation variants with normalization methods. Specifically, from 32 such combinations (i.e., 4 measures $\times$ 8 normalizations), we report only those resulted in an average accuracy higher than the one achieved by Lorentzian (with z-score followed by UnitLength), the new state-of-the-art lock-step distance measure based on our previous analysis (Section 5). (Refer to [73] for raw numbers and detailed pairwise analysis.)

In addition to these pairwise comparisons, we also evaluate the significance of the differences when considered all together. Figure 4 shows the average rank across datasets of five combinations of NCC_c with normalization methods. Similarly to the pairwise analysis, we observe that combinations with z-score, MeanNorm, and

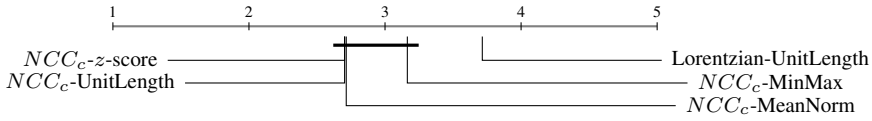


Figure 4: Ranking of different normalization methods for NCC_c based on the average of their ranks across datasets, using Lorentzian with UnitLength as the baseline method.

UnitLength normalizations lead to significant improvements according to the Friedman test followed by a post-hoc Nemenyi test to assess the significance of the differences in the ranking. Combinations of NCC_c with AdaptiveScaling or MinMax do not achieve significant improvement. We observe that both statistical evaluation approaches lead to similar conclusions.

For completeness, we report another analysis using ED as the baseline instead of the Lorentzian distance (we omit the figure due to space limitation). NCC_c in combination with z-score, UnitLength, and MeanNorm normalization methods outperform ED but, in contrast to Figure 4, now combinations with AdaptiveScaling and MinMax are also significantly better than ED. This analysis confirms our results in Section 5 that the Lorentzian distance (and other L_1 variants) are more powerful than ED. In addition, our analysis indicates that NCC_c outperforms all lock-step measures with all different normalizations, making it a strong baseline method for time-series comparison.

We now turn our focus to elastic measures and their performance against sliding measures.

7 Time-Series Elastic Measures

In this section, we study 7 elastic measures, a popular category of measures for time-series comparison.

As discussed earlier, sliding measures find a global alignment by sliding one time series against the other. In contrast, elastic measures create a non-linear mapping between time-series data points to support flexible alignment of different regions. Through this mapping, elastic measures permit time series to "stretch" or "shrink" their observations to improve time-series matching. Most elastic measures rely on dynamic programming to find this mapping efficiently by defining recursive formulas over a m-by-m matrix M that contains in each cell the ED (or some other lock-step measure) between every point of one time series against every point of another time series. In general, the goal of different elastic measures in the literature is to employ different strategies to find a warping path, $W = \{w_1, \ldots, w_k\}$, with $k \geq m$, a contiguous set of matrix cells that shows the mapping of every point of one time series to one, more, or none of the points of the other time series. To improve the efficiency and the accuracy of elastic measures, it is a common practice to introduce constraints (i.e., parameters) to guide the warping path to visit only a subset of cells in M.

The first elastic measure, DTW [87, 88], was proposed as a speech recognition tool and, later, it was introduced in the time-series literature as a suitable approach for time-series comparison [8]. DTW finds the warping path that minimizes the distances between all data points. In the original form, DTW is parameter-free, however, many approaches have been proposed to define bands (i.e., the shape of the subset cells of matrix M that the warping path is permitted to visit) and the width or window (i.e., size) of the bands. We use the Sakoe-Chiba band [88], which is the most frequently used in practice [29], and we tune the window δ using parameters shown in Table 4 of [73]. For example, a value $\delta = 10$ indicates a window size 10% of the time-series length.

The Longest Common Subsequence (LCSS) distance is another type of elastic measure that was derived from the idea of edit-distances for characters. Specifically, LCSS introduces a parameter ϵ that serves as a threshold to determine when two points of time series should match [4, 99]. Similarly to DTW, LCSS also constrains the warping window by introducing an additional parameter δ [99]. Edit Distance on Real sequence (EDR) distance [17] is another edit-distance-based measure that similarly to LCSS, uses a parameter ϵ to quantify the distance of points as 0 or 1. EDR also introduces penalties for gaps between matched subsequences. Edit Distance with Real

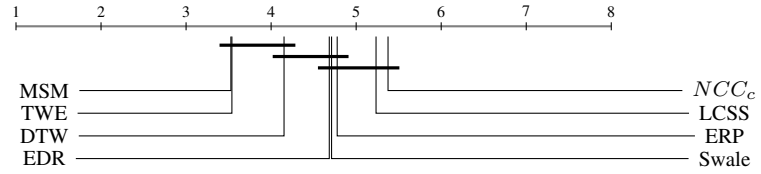


Figure 5: Ranking of elastic and sliding distance measures based on the average of their ranks across datasets, using supervised tuning for their parameters.

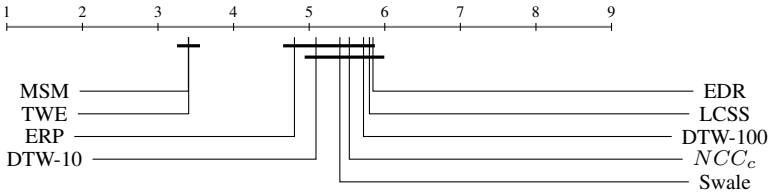


Figure 6: Ranking of elastic and sliding distance measures based on the average of their ranks across datasets, using unsupervised tuning for their parameters.

Penalty (ERP) distance [16] bridges DTW and EDR distance measures by more carefully computing the distance between gaps.

Differently than the previous approaches, the Sequence Weighted Alignment model (Swale) [61] proposes a model to compute the similarity of time series using rewards for matching points and penalties for gaps. Apart from a threshold ϵ parameter, Swale also requires parameters for the reward r and the penalty p. The Move–split–merge (MSM) distance [95] is another elastic measure based on edit-distance but in contrast to DTW, LCSS, and EDR, MSM is a metric. MSM uses a set of operations to replace, insert, or delete values in time series to improve their matching. Finally, Time Warp Edit (TWE) distance [58] is a measure that combines merits from LCSS and DTW. TWE introduces a stiffness parameter ν to control the warping but at the same point it also penalizes matched points.

Evaluation of elastic vs. sliding measures: With the introduction of the 7 elastic measures we are now in position to evaluate their performance against sliding measures, an experiment that has been omitted in all previous studies [6, 29]. Refer to [73] for detailed raw numbers and pairwise comparisons under supervised and unsupervised settings.

To understand the performance of elastic measures against NCC_c, we evaluate the significance of the differences when considered all together. Specifically, Figure 5 shows the average ranks of the elastic measures in the supervised setting and Figure 6 shows the average ranks in the unsupervised setting. We observe that even under supervised settings, 4 out of the 7 elastic measures, namely, LCSS, ERP, EDR, and Swale, do not achieve significantly better performance than NCC_c. The results for MSM, TWE, and DTW, are consistent in both statistical evaluations. For the unsupervised setting, both statistical evaluation approaches agree to an extent. In particular, Figure 6 shows clearly that MSM and TWE outperform NCC_c. However, the remaining 5 elastic measures perform similarity to NCC_c. To validate our findings, we repeat the analysis (we omit figures due to space limitation) and evaluate the significance of the differences when we consider all elastic measures together (i.e., excluding NCC_c). Specifically, we observe that Swale, ERP, EDR, and LCSS do not outperform DTW-10 with statistically significant difference. Interestingly, the supervised LCSS is slightly worse than the unsupervised DTW-10. ERP, which under pairwise evaluation appears to significantly outperform DTW-10, when all measures are considered together, both appear to achieve comparable performance. MSM, TWE, and DTW also perform similarly and all three supervised measures outperform DTW-10. However, under unsupervised settings, MSM and TWE significantly outperform all elastic measures.

Debunking $\mathcal{M}3$ and $\mathcal{M}4$: Our comprehensive evaluation shows clear evidence that sliding measures are strong baselines that most elastic measures do not manage to outperform either in supervised or unsupervised settings,

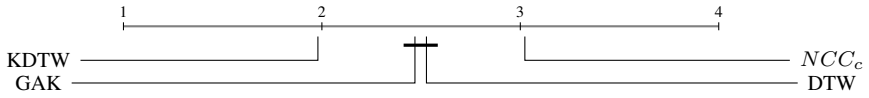


Figure 7: Ranking of kernel measures based on the average of their ranks across datasets (supervised tuning).

which debunks $\mathcal{M}3$. Specifically, from all 5 elastic measures evaluated in the decade-old study [29], namely, LCSS, Swale, EDR, ERP, and DTW, only DTW significantly outperforms cross-correlation under the supervised scenario. In the unsupervised setting, none of the 5 measures outperforms cross-correlation and, interestingly, several of them perform slightly worse. This is a remarkable finding, showing that the simplest type of alignment between time series is very effective and it should have served as a baseline method for elastic measures. Only MSM and TWE, two measures that appeared after [29] show promising results and outperform cross-correlation with statistically significant differences in both supervised and unsupervised settings. Importantly, MSM is the only method that significantly outperforms DTW under supervised settings (according to Wilcoxon) and, under unsupervised settings, both MSM and TWE significantly outperform DTW (with both statistical tests validating this result). Therefore, there is clear evidence that the widely popular DTW is no longer the best elastic distance measure, which debunks $\mathcal{M}4$.

8 Time-Series Kernel Measures

Until now, our analysis focused on three categories of distance measures, namely, lock-step, sliding, and elastic measures, with the goal to provide answers to the four-long standing misconceptions that we discussed in Section 2. Recently, kernel functions [91, 92], a different category of similarity measures, have started to receive attention due to their competitive performance [1]. In contrast to all previously described measures, kernel functions must satisfy the positive semi-definiteness property (p.s.d) [90]. The precise definition is out of the scope of this work (we refer the reader to recent papers for a detailed review [1, 67]) but in simple terms, a function is p.s.d. if the similarity matrix, which contains all pairwise similarity values, has positive eigenvalues. This important property results in convex solutions for several learning tasks involving kernels [21]. In this section, we study 4 representative kernel functions and evaluate their performance against sliding and elastic measures.

Specifically, the first kernel we consider is the Radial Basis Function (RBF) [22], a general purpose kernel function that internally exploits ED but maps data into a high-dimensional space where their separation is easier. To capture similarities between the shifted versions of time series, [100] proposed a sliding kernel to consider all possible alignments between time-series. We include a recently proposed variant of this kernel, namely, SINK, that has achieved competitive results to NCC_c and DTW [67]. Finally, we include two elastic kernel functions, the Global Alignment Kernel (GAK) [23] and Dynamic Time Warping Kernel (KDTW) [59].

Evaluation of kernel functions: Having introduced the 4 kernel functions, we are now in position to evaluate their performance against sliding and elastic measures. As before, we consider both supervised and unsupervised settings. In the supervised setting, we observe that all kernel functions significantly outperform NCC_c with the exception of RBF, which is significantly worse. In the unsupervised settings, KDTW and GAK significantly outperform NCC_c, as before, but SINK achieves comparable performance without outperforming NCC_c. To better understand the performance of KDTW and GAK, which appear to be the strongest kernel functions, we also evaluate the significance of the differences when considered together with all elastic and sliding measures. Figure 7 presents the results for supervised settings and Figure 8 for unsupervised settings. We have omitted elastic measures that based on the earlier analysis did not show competitive results. We observe that GAK achieves comparable performance to DTW under both settings. However, KDTW, significantly outperforms DTW in both unsupervised and superivsed settings. This is in contrast to TWE and MSM measures that were significantly better only under the unsupervised settings. To the best of our knowledge, this is the first time that a kernel function is reported to outperform DTW in both settings.

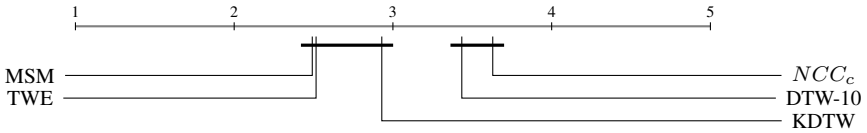


Figure 8: Ranking of kernel measures based on the average of their ranks across datasets (unsupervised tuning).

9 Time-Series Embedding Measures

Previously, we studied approaches that directly exploit a kernel function or a distance measure to compare time series. In this section, we study 4 embedding measures, which are alternative approaches that employ a similarity measure only to construct new representations [1]. These representations are similarity-preserving as the comparison of two representations with ED approximates the comparison of the corresponding original time series with the employed similarity measure.

We consider 4 approaches to construct embedding measures (i.e., ED over learned representations). Specifically, we consider the Generic RepresentAtIon Learning (GRAIL) framework, which employs the SINK kernel [67], the Shift-invariant Dictionary Learning (SIDL) method, which preserves alignment between time series [111], the Similarity Preserving Representation Learning method (SPIRAL), which employs DTW [50], and the Random Warping Series (RWS), which preserves the GAK kernel [104].

Evaluation of embedding measures: For all approaches, we follow [67] and tune required parameters using the recommended values from their corresponding papers. We construct representations of same length (100) for fairness. We observe that GRAIL, is the only framework that constructs robust representations that when ED is used for comparison (under the 1-NN settings), it achieves similar performance to NCC_c, but without significant difference. All other embedding measures perform significantly worse and none of the embedding measures outperform DTW (see detailed raw numbers in [73]. We note, however, that embedding measures (as well as kernel methods), achieve much higher accuracy under different evaluation frameworks (e.g., with SVM classifiers), as shown in [67].

10 Accuracy-to-runtime Analysis

Until now, we have extensively evaluated distance measures based on their accuracy results. However, it is also important to understand the cost associated with each one of these distance measures. In Figure 9, we summarize the accuracy-to-runtime performance of the most prominent measures. The runtime performance includes only inference time (i.e., evaluation on the testing sets). We observe that ED, and all other lock-step measures (omitted), are the fastest but achieve relatively low accuracy (all these measures have $\mathcal{O}(m)$ runtime cost). NCC_c [68] and SINK [67], two methods that rely on the classic cross-correlation measure, provide an excellent trade-off between runtime and accuracy in comparison to ED (these measures have $\mathcal{O}(m \log m)$ runtime cost). We also observe that all other elastic or kernel methods require substantially higher runtime costs to achieve comparable accuracy results to NCC_c (these measures have $\mathcal{O}(m^2)$ runtime cost). In particular, only MSM and TWE significantly outperform NCC_c (see Figure 6) but require two orders of magnitude higher runtime cost. Instead, embedding measures, such as GRAIL [67], show great promise as they can achieve high accuracy without sacrificing runtime performance.

11 Accelerating Elastic Measures

Despite their promise, elastic distance measures scale quadratically to the length of the time series, as noted earlier. Compared to ED, which has linear complexity, elastic distance measures incur an additional runtime overhead, often between one to three orders of magnitude (see Figure 9). This cost would prevent applications from using

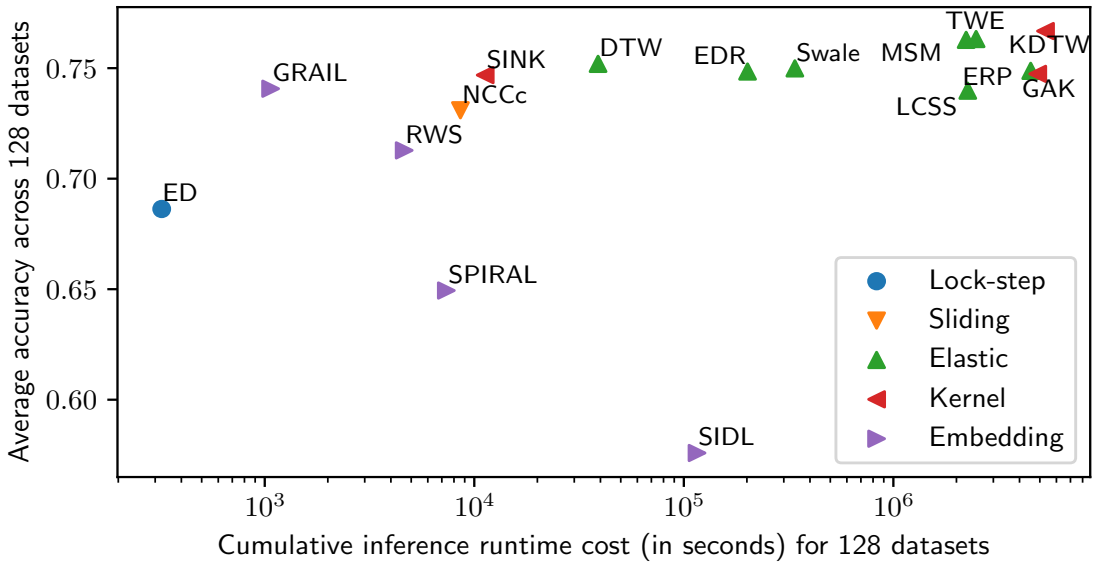


Figure 9: Accuracy-to-runtime comparison.

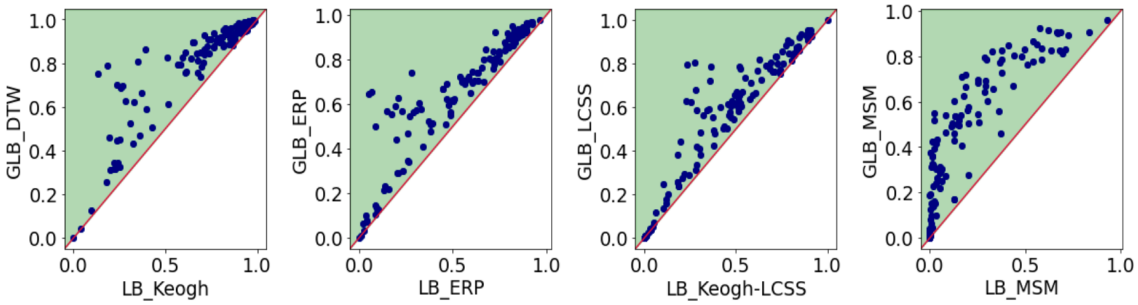


Figure 10: Comparison of the pruning power of GLB variants against state-of-the-art LBs of several popular elastic measures over 128 datasets. The blue dots above the diagonal indicate datasets over which GLB outperforms the state of the art.

elastic measures in large-scale settings. To alleviate this issue, the idea of lower bounding was developed to filter out unpromising candidates before carrying out the expensive elastic distance measure computation [34, 44, 46]. In simple terms, a lower bound (LB) is a fast distance measure that approximates an expensive elastic distance measure and is computed over some summaries of the time series instead of the actual time series.

A plethora of LBs have been developed for elastic distance measures [16, 44, 46, 51, 93, 97, 98], with the goal to improve their pruning power (i.e., *tightness* of LB). Unfortunately, the research effort on LBs has been disproportionally concentrated on Dynamic Time Warping (DTW) [87, 88], which is the oldest elastic measure with at least eight established LBs (see [78] for details). In contrast, newer and better-performing elastic distance measures, such as MSM and TWE, have received little attention, and their LBs are performing poorly. Unfortunately, developing LBs is a challenging task. It is unsustainable to expect a similar research effort for each elastic measure. For this reason, a generalized framework, namely GLB, was recently proposed [78] to accumulate the knowledge from previously developed LBs and eliminate the need for designing separate LBs for each elastic measure. Specifically, GLB outperforms all established LBs across different elastic measures. Figure 10 shows the improvement in pruning power (i.e., the percentage of the true distance computation avoided) achieved by GLB for several popular elastic measures (more details in [78]). Considering that MSM and TWE are the new state-of-the-art elastic measures, we note that GLB accelerates MSM up to 10× and TWE up to 26× in an extensive analysis we performed across 128 datasets [25].

12 Future Directions

With the new knowledge in place, several new challenges open that we hope to spark new research directions. Below, we provide three areas that we believe require more attention and can potentially lead to substantial improvements in the entire area of time-series similarity search:

- Identifying more accurate normalizations. Our work was the first to study the performance of 8 normalization methods. We identified multiple distance measures outperforming the previous SOTA measures only when combined with appropriate normalization methods. In our view, inventing a new normalization method that achieves significant accuracy improvements by preprocessing data differently and without changing existing methods and systems would be a breakthrough.
- Tuning parameters, or selecting appropriate distance measures per dataset in an unsupervised manner. Unfortunately, there are no principled methodologies currently for selecting distance measures or tuning their parameters, despite significant recent attention in AutoML for other domains.
- Improving and evaluating the performance of embedding measures. These measures show the most promise based on their runtime-to-accuracy trade-off. To the best of our knowledge and based on our comprehensive study, there are no embedding measures that significantly outperform the most vigorous elastic measures in terms of accuracy. Recent advances in deep neural networks [101] may lead to embeddings that substantially outperform elastic measures.

13 Conclusion

We presented a comprehensive evaluation to validate the performance of 71 distance measures. Our study debunked four long-standing misconceptions in the time-series literature and established new state-of-the-art results for lock-step, sliding, elastic, kernel, and embedding measures. Our findings prepare the ground for the development of distance measures with implications across time-series analytical tasks. Importantly, our work has implications for general-purpose similarity search problems over high-dimensional data. For example, several similarity search methodologies rely heavily on the concepts of lower bounding to prune unnecessary comparisons [32, 76]. Similarly to how GLB abstracted the costs of different elastic measures and generalized lower bounds for time series, we believe a similar concept can be applied in the case of lock-step measures (e.g., Euclidean distance) and the corresponding data summarization methods. In addition, our work identified lock-step measures that outperform Euclidean distance and lock-step measures performing exceptionally well only under certain normalizations. However, the literature in the similarity search area has largely focused on developing methods assuming Euclidean distance is the underlying distance measure. Our work may lead to new solutions for the new, better-performing distance measures. Finally, the methodologies presented for constructing embedding measures are sufficiently generic and can complement solutions focusing on learning embeddings from data [101] (e.g., concatenate deep embeddings with our similarity-preserving embeddings or improve deep embeddings by integrating our similarity-preserving embeddings in the loss functions).

Acknowledgments: We thank Kaize Wu for his help and useful discussions. This research was supported in part by a Google DAPA Research Award, gifts from NetApp, Cisco Systems, and Exelon Utilities, and an NSF Award CCF-1139158. Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation.

References

[1] Amaia Abanda, Usue Mori, and Jose A Lozano. A review on distance based time series classification. Data Mining and Knowledge Discovery, 33(2):378–412, 2019.

[2] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient similarity search in sequence databases. In FODO, pages 69–84, 1993.

[3] Rakesh Agrawal, King-Ip Lin, Harpreet S. Sawhney, and Kyuseok Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In Proceeding of the 21th International Conference on Very Large Data Bases, pages 490–501. Citeseer, 1995.

[4] Henrik André-Jönsson and Dushan Z Badal. Using signature files for querying time-series data. In European Symposium on Principles of Data Mining and Knowledge Discovery, pages 211–220. Springer, 1997.

[5] Johannes Aßfalg, Hans-Peter Kriegel, Peer Kröger, Peter Kunath, Alexey Pryakhin, and Matthias Renz. Similarity search on time series based on threshold queries. In International Conference on Extending Database Technology, pages 276–294. Springer, 2006.

[6] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery, 31(3):606–660, 2017.

[7] Mohini Bariya, Alexandra von Meier, John Paparrizos, and Michael J Franklin. k-shapestream: Probabilistic streaming clustering for electric grid events. In 2021 IEEE Madrid PowerTech, pages 1–6. IEEE, 2021.

[8] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In AAAI Workshop on KDD, pages 359–370, 1994.

[9] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J Franklin. Sand: streaming subsequence anomaly detection. Proceedings of the VLDB Endowment, 14(10):1717–1729, 2021.

[10] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J Franklin. Sand in action: subsequence anomaly detection for streams. Proceedings of the VLDB Endowment, 14(12):2867–2870, 2021.

[11] Paul Boniol, John Paparrizos, Yuhao Kang, Themis Palpanas, Ruey S Tsay, Aaron J Elmore, and Michael J Franklin. Theseus: navigating the labyrinth of time-series anomaly detection. Proceedings of the VLDB Endowment, 15(12):3702–3705, 2022.

[12] Paul Boniol, John Paparrizos, and Themis Palpanas. New trends in time series anomaly detection. In Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023, pages 847–850. OpenProceedings.org, 2023. doi: 10.48786/edbt.2023.80. URL https://doi.org/10.48786/edbt.2023.80.

[13] R Bracewell. Pentagram notation for cross correlation. the fourier transform and its applications. New York: McGraw-Hill, 46: 243, 1965.

[14] Lisa Gottesfeld Brown. A survey of image registration techniques. ACM computing surveys (CSUR), 24(4):325–376, 1992.

[15] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. City, 1(2):1, 2007.

[16] Lei Chen and Raymond Ng. On the marriage of Lp-norms and edit distance. In VLDB, pages 792–803, 2004.

[17] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In SIGMOD, pages 491–502, 2005.

[18] Yueguo Chen, Mario A Nascimento, Beng Chin Ooi, and Anthony KH Tung. Spade: On shape-based pattern detection in streaming time series. In ICDE, pages 786–795, 2007.

[19] Kelvin Kam Wing Chu and Man Hon Wong. Fast time-series searching with scaling and shifting. In Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 237–248. Citeseer, 1999.

[20] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation, 19(90):297–301, 1965.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. Machine learning, 20(3):273–297, 1995.

[22] Nello Cristianini and John Shawe-Taylor. An introduction to support vector machines and other kernel-based learning methods. Cambridge university press, 2000.

[23] Marco Cuturi. Fast global alignment kernels. In Proceedings of the 28th international conference on machine learning (ICML-11), pages 929–936, 2011.

[24] Michele Dallachiesa, Themis Palpanas, and Ihab F Ilyas. Top-k nearest neighbor search in uncertain data series. Proceedings of the VLDB Endowment, 8(1):13–24, 2014.

[25] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. The ucr time series classification archive, October 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[26] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. The Journal of Machine Learning Research, 7:1–30, 2006.

[27] Michel-Marie Deza and Elena Deza. Dictionary of distances. Elsevier, 2006.

[28] Michel Marie Deza and Elena Deza. Encyclopedia of distances. In Encyclopedia of distances, pages 1–583. Springer, 2009.

[29] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. Proceedings of the VLDB Endowment, 1(2):1542–1552, 2008.

[30] Alejandro Domínguez. A history of the convolution operation [retrospectroscope]. IEEE pulse, 6(1):38–49, 2015.

[31] Adam Dziedzic, John Paparrizos, Sanjay Krishnan, Aaron Elmore, and Michael Franklin. Band-limited training and inference for convolutional neural networks. In International Conference on Machine Learning, pages 1745–1754. PMLR, 2019.

[32] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. Proceedings of the VLDB Endowment, 12(2):112–127, 2018.

[33] Philippe Esling and Carlos Agon. Time-series data mining. ACM Computing Surveys (CSUR), 45(1):12, 2012.

[34] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In SIGMOD, pages 419–429, 1994.

[35] Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. Index-based most similar trajectory search. In ICDE, pages 816–825, 2007.

[36] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association, 32:675–701, 1937.

[37] Daniel G Gavin, W Wyatt Oswald, Eugene R Wahl, and John W Williams. A statistical approach to evaluating distance metrics and analog assignments for pollen records. Quaternary Research, 60(3):356–367, 2003.

[38] Rafael Giusti and Gustavo EAPA Batista. An empirical comparison of dissimilarity measures for time series classification. In BRACIS, pages 82–88, 2013.

[39] Rahul Goel, Sandeep Soni, Naman Goyal, John Paparrizos, Hanna Wallach, Fernando Diaz, and Jacob Eisenstein. The social dynamics of language change in online networks. In Social Informatics: 8th International Conference, SocInfo 2016, Bellevue, WA, USA, November 11-14, 2016, Proceedings, Part I 8, pages 41–57. Springer, 2016.

[40] Dina Q Goldin and Paris C Kanellakis. On similarity queries for time-series data: constraint specification and implementation. In International Conference on Principles and Practice of Constraint Programming, pages 137–153. Springer, 1995.

[41] Hao Jiang, Chunwei Liu, Qi Jin, John Paparrizos, and Aaron J Elmore. Pids: attribute decomposition for improved compression and query performance in columnar storage. Proceedings of the VLDB Endowment, 13(6):925–938, 2020.

[42] Hao Jiang, Chunwei Liu, John Paparrizos, Andrew A Chien, Jihong Ma, and Aaron J Elmore. Good to the last bit: Data-driven encoding with codecdb. In Proceedings of the 2021 International Conference on Management of Data, pages 843–856, 2021.

[43] Eamonn Keogh and Jessica Lin. Clustering of time-series subsequences is meaningless: Implications for previous and future research. Knowledge and Information Systems, 8(2):154–177, 2005.

[44] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. Knowledge and Information Systems, 7(3):358–386, 2005.

[45] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In SIGMOD, pages 151–162, 2001.

[46] Sang-Wook Kim, Sanghyun Park, and Wesley W Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In Data Engineering, 2001. Proceedings. 17th International Conference on, pages 607–614. IEEE, 2001.

[47] Sanjay Krishnan, Aaron J Elmore, Michael Franklin, John Paparrizos, Zechao Shang, Adam Dziedzic, and Rui Liu. Artificial intelligence in resource-constrained and shared environments. ACM SIGOPS Operating Systems Review, 53(1):1–6, 2019.

[48] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10):1995, 1995.

[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. nature, 521(7553):436, 2015.

[50] Qi Lei, Jinfeng Yi, Roman Vaculin, Lingfei Wu, and Inderjit S Dhillon. Similarity preserving representation learning for time series analysis. arXiv preprint arXiv:1702.03584, 2017.

[51] Daniel Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. Pattern recognition, 42(9):2169–2180, 2009.

[52] Chung-Sheng Li, Philip S. Yu, and Vittorio Castelli. Hierarchyscan: A hierarchical similarity search algorithm for databases of long sequences. In ICDE, pages 546–553. IEEE, 1996.

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

[54] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. Data Mining and Knowledge Discovery, 29(3):565–592, 2015.

[55] Chunwei Liu, Hao Jiang, John Paparrizos, and Aaron J Elmore. Decomposed bounded floats for fast compression and queries. Proceedings of the VLDB Endowment, 14(11):2586–2598, 2021.

[56] Shinan Liu, Tarun Mangla, Ted Shaowang, Jinjin Zhao, John Paparrizos, Sanjay Krishnan, and Nick Feamster. Amir: Active multimodal interaction recognition from video and network traffic in connected environments. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 7(1):1–26, 2023.

[57] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth. Machine learning for internet of things data analysis: A survey. Digital Communications and Networks, 2017.

[58] Pierre-François Marteau. Time warp edit distance with stiffness adjustment for time series matching. IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(2):306–318, 2008.

[59] Pierre-François Marteau and Sylvie Gibet. On recursive edit distance kernels with application to time series classification. IEEE transactions on neural networks and learning systems, 26(6):1121–1133, 2014.

[60] Kathy McKeown, Hal Daume III, Snigdha Chaturvedi, John Paparrizos, Kapil Thadani, Pablo Barrio, Or Biran, Suvarna Bothe, Michael Collins, Kenneth R Fleischmann, et al. Predicting the impact of scientific concepts using full-text features. Journal of the Association for Information Science and Technology, 67(11):2684–2696, 2016.

[61] Michael D Morse and Jignesh M Patel. An efficient and accurate method for evaluating time series similarity. In SIGMOD, pages 569–580, 2007.

[62] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and Brandon Westover. Exact discovery of time series motifs. In Proceedings of the 2009 SIAM international conference on data mining, pages 473–484. SIAM, 2009.

[63] Peter Nemenyi. Distribution-free Multiple Comparisons. PhD thesis, Princeton University, 1963.

[64] Themis Palpanas. Data series management: the road to big sequence analytics. ACM SIGMOD Record, 44(2):47–52, 2015.

[65] Ioannis Paparrizos. Fast, scalable, and accurate algorithms for time-series analysis. PhD thesis, Columbia University, 2018.

[66] John Paparrizos. 2018 ucr time-series archive: Backward compatibility, missing values, and varying lengths, January 2019. https://github.com/johnpaparrizos/UCRArchiveFixes.

[67] John Paparrizos and Michael J Franklin. Grail: efficient time-series representation learning. Proceedings of the VLDB Endowment, 12(11):1762–1777, 2019.

[68] John Paparrizos and Luis Gravano. k-shape: Efficient and accurate clustering of time series. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pages 1855–1870. ACM, 2015.

[69] John Paparrizos and Luis Gravano. Fast and accurate time-series clustering. ACM Transactions on Database Systems (TODS), 42 (2):8, 2017.

[70] John Paparrizos and Sai Prasanna Teja Reddy. Odyssey: An engine enabling the time-series clustering journey. Proceedings of the VLDB Endowment, 16(12):4066–4069, 2023.

[71] John Paparrizos, Ryen W White, and Eric Horvitz. Detecting devastating diseases in search logs. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 559–568, 2016.

[72] John Paparrizos, Ryen W White, and Eric Horvitz. Screening for pancreatic adenocarcinoma using signals from web search logs: Feasibility study and results. Journal of oncology practice, 12(8):737–744, 2016.

[73] John Paparrizos, Chunwei Liu, Aaron J Elmore, and Michael J Franklin. Debunking four long-standing misconceptions of time-series distance measures. In Proceedings of the 2020 ACM SIGMOD international conference on management of data, pages 1887–1905, 2020.

[74] John Paparrizos, Chunwei Liu, Bruno Barbarioli, Johnny Hwang, Ikraduya Edian, Aaron J Elmore, Michael J Franklin, and Sanjay Krishnan. Vergedb: A database for iot analytics on edge devices. In CIDR, 2021.

[75] John Paparrizos, Paul Boniol, Themis Palpanas, Ruey S Tsay, Aaron Elmore, and Michael J Franklin. Volume under the surface: a new accuracy evaluation measure for time-series anomaly detection. Proceedings of the VLDB Endowment, 15(11):2774–2787, 2022.

[76] John Paparrizos, Ikraduya Edian, Chunwei Liu, Aaron J Elmore, and Michael J Franklin. Fast adaptive similarity search through variance-aware quantization. In 2022 IEEE 38th International Conference on Data Engineering (ICDE), pages 2969–2983. IEEE, 2022.

[77] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection. Proceedings of the VLDB Endowment, 15(8):1697–1711, 2022.

[78] John Paparrizos, Kaize Wu, Aaron Elmore, Christos Faloutsos, and Michael J Franklin. Accelerating similarity search for elastic measures: A study and new generalization of lower bounding distances. Proceedings of the VLDB Endowment, 16(8):2019–2032, 2023.

[79] Athanasios Papoulis. The Fourier integral and its applications. McGraw-Hill, 1962.

[80] François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. Pattern Recognition, 44(3):678–693, 2011.

[81] François Petitjean, Germain Forestier, Geoffrey I Webb, Ann E Nicholson, Yanping Chen, and Eamonn Keogh. Dynamic time warping averaging of time series allows faster and more accurate classification. In 2014 IEEE international conference on data mining, pages 470–479. IEEE, 2014.

[82] François Petitjean, Germain Forestier, Geoffrey I Webb, Ann E Nicholson, Yanping Chen, and Eamonn Keogh. Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. Knowledge and Information Systems, 47(1):1–26, 2016.

[83] Davood Rafiei and Alberto Mendelzon. Similarity-based queries for time series data. In ACM SIGMOD Record, volume 26, pages 13–25. ACM, 1997.

[84] Chotirat Ann Ralanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michail Vlachos, and Gautam Das. Mining time series data. In Data mining and knowledge discovery handbook, pages 1069–1103. Springer, 2005.

[85] Chotirat Ann Ratanamahatana and Eamonn Keogh. Making time-series classification more accurate using learned constraints. In SDM, pages 11–22, 2004.

[86] John Rice. Mathematical statistics and data analysis. Cengage Learning, 2006.

[87] Hiroaki Sakoe and Seibi Chiba. A dynamic programming approach to continuous speech recognition. In ICA, pages 65–69, 1971.

[88] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. IEEE transactions on acoustics, speech, and signal processing, 26(1):43–49, 1978.

[89] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. Braid: Stream mining through group lag correlations. In SIGMOD, pages 599–610. ACM, 2005.

[90] Bernhard Schölkopf and Alexander J Smola. Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press, 2002.

[91] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In International Conference on Artificial Neural Networks, pages 583–588. Springer, 1997.

[92] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. Neural computation, 10(5):1299–1319, 1998.

[93] Yilin Shen, Yanping Chen, Eamonn Keogh, and Hongxia Jin. Accelerating time series searching with large uniform scaling. In Proceedings of the 2018 SIAM International Conference on Data Mining, pages 234–242. SIAM, 2018.

[94] Jin Shieh and Eamonn Keogh. i sax: indexing and mining terabyte sized time series. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 623–631. ACM, 2008.

[95] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. The move-split-merge metric for time series. TKDE, 25(6):1425–1438, 2013.

[96] Emmanouil Sylligardos, Paul Boniol, John Paparrizos, Panos Trahanias, and Themis Palpanas. Choose wisely: An extensive evaluation of model selection for anomaly detection in time series. Proceedings of the VLDB Endowment, 16(11):3418–3432, 2023.

[97] Chang Wei Tan, François Petitjean, and Geoffrey I Webb. Elastic bands across the path: A new framework and method to lower bound dtw. In Proceedings of the 2019 SIAM International Conference on Data Mining, pages 522–530. SIAM, 2019.

[98] Chang Wei Tan, François Petitjean, and Geoffrey I Webb. Fastee: Fast ensembles of elastic distances for time series classification. Data Mining and Knowledge Discovery, 34(1):231–272, 2020.

[99] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In Proceedings 18th international conference on data engineering, pages 673–684. IEEE, 2002.

[100] Gabriel Wachman, Roni Khardon, Pavlos Protopapas, and Charles R Alcock. Kernels for periodic time series arising in astronomy. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 489–505. Springer, 2009.

[101] Qitong Wang and Themis Palpanas. Seanet: A deep learning architecture for data series similarity search. IEEE Transactions on Knowledge and Data Engineering, 2023.

[102] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. Experimental comparison of representation methods and distance measures for time series data. Data Mining and Knowledge Discovery, pages 1–35, 2013.

[103] Frank Wilcoxon. Individual comparisons by ranking methods. Biometrics Bulletin, pages 80–83, 1945.

[104] Lingfei Wu, Ian En-Hsu Yen, Jinfeng Yi, Fangli Xu, Qi Lei, and Michael Witbrock. Random warping series: A random features method for time-series embedding. In AISTATS, pages 793–802, 2018.

[105] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In Proceedings of the 23rd international conference on Machine learning, pages 1033–1040. ACM, 2006.

[106] Jaewon Yang and Jure Leskovec. Patterns of temporal variation in online media. In WSDM, pages 177–186, 2011.

[107] Lexiang Ye and Eamonn Keogh. Time series shapelets: a new primitive for data mining. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 947–956. ACM, 2009.

[108] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In 2016 IEEE 16th international conference on data mining (ICDM), pages 1317–1322. IEEE, 2016.

[109] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. VLDB, 2000.

[110] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. Similarity search: the metric space approach, volume 32. Springer Science & Business Media, 2006.

[111] Guoqing Zheng, Yiming Yang, and Jaime Carbonell. Efficient shift-invariant dictionary learning. In SIGKDD, pages 2095–2104. ACM, 2016.

[112] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. Ads: the adaptive data series index. The VLDB Journal—The International Journal on Very Large Data Bases, 25(6):843–866, 2016.

Recent Approaches and Trends in Approximate Nearest Neighbor Search, with Remarks on Benchmarking

Martin Aumüller[†], Matteo Ceccarello[*]
[†]IT University of Copenhagen, maau@itu.dk
*University of Padova, matteo.ceccarello@unipd.it

Abstract

Nearest neighbor search is a computational primitive whose efficiency is paramount to many applications. As such, the literature recently blossomed with many works focusing on improving its effectiveness in an approximate setting. In this overview paper, we review recent advances of the state of the art and discuss some trends. Given the practical relevance of the problem, new approaches need to be thoroughly benchmarked. We therefore review some recent benchmarking efforts and provide advice on the benchmarking pipeline.

1 Introduction

Nearest neighbor search is a key component in many computer science applications. For example, using CLIP embeddings [50] both images and text can be mapped to dense vectors in a vector space; retrieving images that match a text description then boils down to embedding the text into a query vector and searching for images whose vectors are closest to the query vector under some distance measure. Using nearest neighbor search, large language models (LLMs) can also be augmented with knowledge that was not present in the training data [36]. Unfortunately, exact nearest neighbor search in high-dimensional data—such as the dense vectors generated by deep learning-based embedding pipelines—is a notoriously difficult problem that usually requires scanning through the whole dataset. This excludes the possibility of scalable approaches for billion-scale datasets that are common in today's applications.

To enable scalable nearest neighbor search, the research focus turned to approximate nearest neighbor search (ANN). In empirical settings, this usually means that an implementation does not provide a guarantee on the quality of the returned vectors. Instead, the user provides—based on knowledge of the data distribution and the query workloads—parameters that are used to build and to search the index data structure, respectively. Given a query point and some search parameters, the index is used to generate a candidate set for the query. The closest vectors among these candidates are returned as the (presumably) nearest neighbors for a given query. The smaller the candidate set, the faster the search, but also the lower the result accuracy. Section 2 gives an overview over general approaches to approximate nearest neighbor search.

ANN-benchmarks [10] presents the state-of-the-art benchmark on million-scale approximate nearest neighbor search implementations. In the benchmarking run published in April 2023, more than 30 implementations were tested on a collection of datasets. Each implementation was run on a single thread. The results for a single dataset are depicted in Figure 1. On a high level, many implementations achieve a throughput of more than 1,000 queries per second at an average recall of at least 90%. In particular, many implementations still perform well in the setting of average recall at least 99%. As a baseline, a bruteforce solution using BLAS achieved a throughput of 16 queries per second. Most of the approaches that perform best implement a graph-based approach with the notable exception of Google's ScaNN [25] and Meta's FAISS [35] which both implement a clustering-based approach as their main data structure. Please see the project website for more details on the used implementations. We review benchmarking efforts and pitfalls in Section 3.

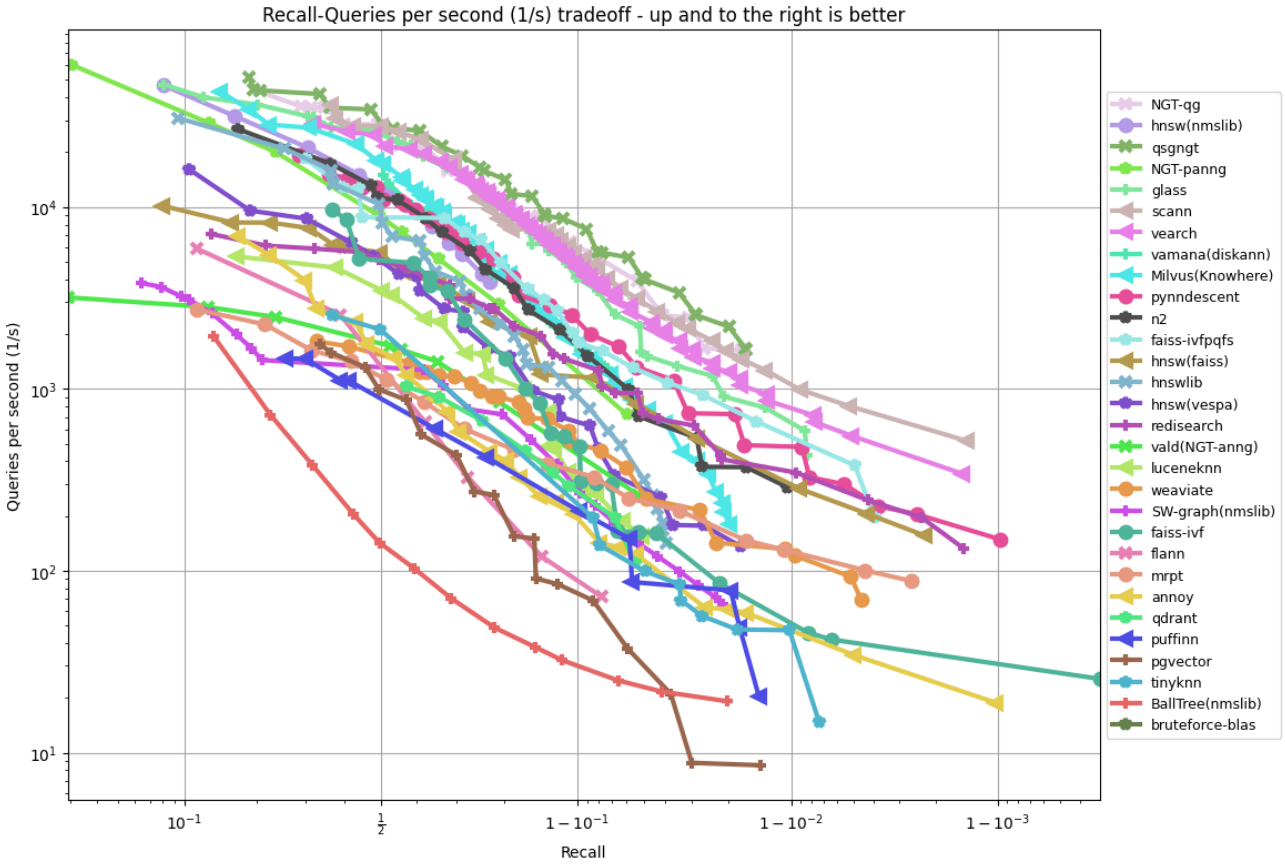


Figure 1: April 2023 run of ANN-benchmarks [10] (`http://ann-benchmarks.com`) on 100-dimensional vectors from GloVe embeddings [47] on a twitter corpus. The x-axis represents the average fraction of true 10-nearest neighbors returned (on a logarithmic scale); the y-axis provides the queries per second achieved, on a logarithmic scale. Both index building and searching is conducted on a single thread. Reported points represent the Pareto-frontier over a grid search on the parameter space. Interactive visualisations are available on the project website. Benchmarking run carried out by E. Bernhardsson.

This overview paper focuses on approaches that are not covered by the current benchmarking efforts. While theoretical breakthroughs have been achieved over the last years, for example by Andoni et al. [5], our focus lies on approaches that are supported by efficient implementations targeted to solve nearest neighbor search on real-world datasets. Section 4 is dedicated to this recent work. Finally, we close the overview by identifying recent trends and promising directions for future work in Section 5.

1.1 Problem setup

Formally, the task in k-nearest neighbor search is defined as follows. Let $(\mathcal{X}, \text{dist})$ be a metric space, and let $k \geq 1$ be an integer. Given a dataset $S \subseteq \mathcal{X}$ of n data points $(p_1, \ldots, p_n) \in \mathcal{X}^n$, the task is to build an index data structure that supports the following queries: Given a query point $q \in \mathcal{X}$, return a sequence $\mathcal{I}_q = (i_1, \ldots, i_k)$ of unique indices of data points in S such that $p_{i_1}, \ldots, p_{i_k}$ minimize the distance to q.

For simplicity, we will focus on the case that $\mathcal{X} = \mathbb{R}^d$, i.e., we consider d-dimensional real-valued vectors. Classical distance metrics are L_p norms, in particular for $p = 2$ (Euclidean distance), and inner product dissimilarity, associated with the task commonly known as maximum inner product search. Other interesting

cases are length-d bitstrings $\mathcal{X} = \{0,1\}^d$ under Hamming distance and collections of sets $\mathcal{X} = \mathcal{P}(U)$ under Jaccard similarity over a finite universe U with $\mathcal{P}(U)$ being the power set of U.

In the case that distances are unique, we can identify by $\mathcal{I}_q^*$ the set of indices of the true k-nearest neighbors of a query point q. As a quality measure, we consider the recall $|\mathcal{I}_q \cap \mathcal{I}_q^*|/k$. We call a method exact if it guarantees a recall of 1, and we call it approximate otherwise. In the context of approximate methods, papers often report on the average recall over a set Q of queries. If distances are not unique, distance-based recall variants are available [10].

2 A general overview over high-dimensional indexing

There exists a plethora of different approaches for solving nearest neighbor search. The most successful approaches can be categorized into *clustering-based*, *graph-based*, *hashing-based*, and *tree-based* approaches. For the notable exception of graph-based approaches, nearest neighbor search is usually solved by partitioning the space $\mathbb{R}^d$ into M disjoint parts $R_1, \ldots, R_M$ such that $\bigcup_{1 \le i \le M} R_i = \mathbb{R}^d$.

2.1 Indexing techniques for high-dimensional data

We provide a short overview of approaches and highlight a well-established method from each category. Each implementation comes with certain parameter choices used during the indexing phase (building the ANN data structure) and the querying phase (searching for the approximate nearest neighbors). To ease the explanation and make an attempt of unifying the landscape, we provide explanations that focus on a single build parameter M and a single search parameter ℓ.

Clustering-based approaches (IVF [35]). Given a dataset $S \subseteq \mathbb{R}^d$ and two parameters M and ℓ, run a clustering algorithm such as M-means to find M centroids. By associating each point with its closest centroid, the space is partitioned into M parts. The data structure that stores the centroids and the associated lists is referred to as an inverted file index (IVF). To find nearest neighbors to a query $q \in \mathbb{R}^d$, inspect the points associated to the ℓ closest centroids of q. Since this itself is a nearest neighbor search task, for large M an index over the centroids is employed. Clustering-based approaches usually provide very compelling index size since each point is stored only once with its associated centroid. The build time of a clustering-based approach is dominated by clustering the data points, which is often done on a sample. The final assignment carries out $O(nM)$ distance computations to centroids if the assignment is exact; as before, this can be sped up by indexing the centroids.

Graph-based approaches (HNSW [43]). Given a dataset $S \subseteq \mathbb{R}^d$ and parameters M, ℓ, the goal is to build a graph $G = (V, E)$, where each point is represented by a vertex and edges exist between a point and a "diverse" set of at most M points. Let us assume that such a graph G is given. To find the nearest neighbors of a query point q, HNSW uses a hierarchy of graphs to find a good entry point into the bottom-layer graph that indexes all points. Given such a start point, carry out a greedy hill climbing. In each round, consider the currently closest point to the query not considered before. Inspect the neighborhood and compute the distances to the query point. After each round, trim the list of current closest points (inspected and non-inspected) to ℓ, which is usually called the beam width. Terminate if all ℓ points have been considered. (Note that this is not a bound on the number of distance computations, since considered points might be trimmed off.) To build the graph, order all the points and insert them one-by-one using the search algorithm, often with a smaller ℓ' than used for the queries. From the points inspected in this search, a pruned set of M points is chosen as neighbors of the inserted point. Additionally, pruning might be necessary for its neighbors if their degree bound M is not met. Graph-based indexes usually provide compelling index sizes when M is small (the number of edges can be as large as Mn). The index build

time is usually rather high for graph-based approaches, since individual searches are carried out for each data point.

Hashing-based approaches (FALCONN [3]) FALCONN is an approach based on locality-sensitive hashing optimized for inner product similarity on unit length vectors. It uses crosspolytope LSH as its LSH function. A crosspolytope LSH function is described by a rotation matrix, which is chosen at random. The hash value of a point is the closest base vector in $\mathbb{R}^d$ when applying the random rotation to the point. Given a set of points S and two parameter M, ℓ, the data structure works as follows. Choose M random LSH functions mapping data points to hash values $\mathcal{R}$, where each function is the concatenation of two to three random crosspolytope LSH functions. Hash each data point M times with independent hash function choices, and store the point in M buckets, one per hash function. These buckets together with the collection of hash functions form the index. Given a query point and collection of M hash tables, hash the query point using the same hash functions and consider the data points that also reside in the bucket as candidates. Traditionally, LSH suffers from large indexes since independent repetitions provide the "best" buckets in the sense that among all buckets, it is most likely to find a close points in the bucket identified by the query hash code. However, if space is a concern, one can use a smaller M value and if less than ℓ points are found, check neighboring buckets using a multiprobing approach. The index build time of a hashing-based approach is usually dominated by hashing each data point M times. Fast evaluation tricks, such as applying the fast Hadamard transform [3] and pooling/tensoring approaches [18] can be employed to lower this cost.

Tree-based approaches (MRPT [30]). MRPT builds a collection of trees based on sparse random projections. Given a set of points S and two parameters M, ℓ, the data structure works as follows. First, a node in a tree is described by a hyperplane a that splits up a point set $S' \subseteq S$. At the root, the whole dataset is taken into consideration, and a leaf is created as soon as the number of points at a node is below a certain threshold. Instead of a single tree, M trees are created to boost the quality of the results. Given a query point and a collection of M trees, carry out root-to-leaf-traversals in each tree for the query. In MRPT, a voting search is carried out by considering a point in a leaf as a candidate if it appears in at least ν different trees. MRPT results in compelling index sizes for small M values since each level of the tree contains a single random hyperplane, only storing the split value in a node. The build time is dominated by finding the individual splits, which typically requires in each node and over all trees, to evaluate the projection value and select the median. [32] describe a method to automatically select hyper-parameters for this approach.

2.2 System Architecture

The standard system architecture for ANN search is depicted in Figure 2. Given a dataset $S \subseteq \mathcal{X}$ and a set of build parameters $\mathcal{P}_{\text{build}}$, an index is built following the approaches mentioned in the previous subsection. Both the build time and the index size is often crucial for the feasibility of an approach. Given a query point $q \in \mathcal{X}$ and a set of query parameters $\mathcal{P}_{\text{query}}$, the index is used to generate the candidate set. This candidate set is usually refined using a quantization or sketching technique that stores small summaries of each data point. The goal of the refinement is to discard candidate points that are unlikely to be among the k nearest neighbors. In a final reranking step, exact distance computations between the refined candidates and the query point are carried out, and the indices of the k points with smallest distance to the query are returned as the answer to the query. In the case that memory resources are sparse, for example when dataset vectors do not fit into main memory, the final re-ranking step might not be carried out, which usually results in a loss in precision.

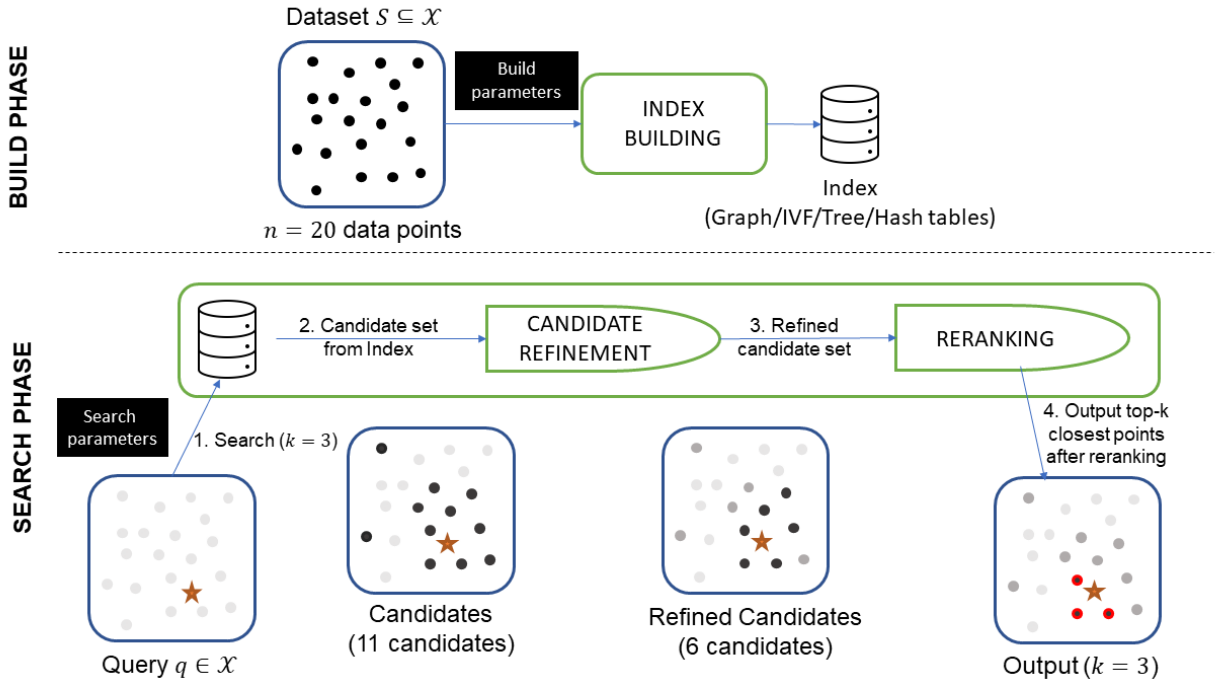


Figure 2: Overview of the phases of a traditional ANN implementation.

3 Benchmarking ANN implementations

Assessing the performance of a new method is at the same time crucial and challenging. The new method itself often has a lot of configurations to explore, and the number of baselines to compare with grows by the day, with each baseline featuring a lot of parameters. In such a scenario, benchmarking efforts such as the ANN-benchmarks [10] mentioned in the introduction— also used for benchmarking in billion-scale settings [53]— and the Lernaean Hydra framework for data series similarity [21] provide a very useful stepping stone.

In particular, a benchmarking infrastructure such as ANN-benchmarks provides a collection of baseline algorithms, along with sensible ranges of their parameters to test. Algorithms are then evaluated according to a standardized evaluation protocol: each approach is first set up with indexing parameters; then it is fed the data to be queried, allowing it to build index structures; finally several query groups are executed on the same index, with different query parameters.

We believe that it is much easier to integrate a new method in an existing benchmarking infrastructure, rather than implementing a custom one for each new paper. We therefore urge the community to adopt shared benchmarking infrastructures in order both to avoid reinventing the wheel and to make results more easily comparable across papers. Even if the core pipeline cannot be used, the data preprocessing (for example, the fixed definition of query workloads) and result postprocessing (for example, the availability of groundtruth data and evaluations scripts) can be used in isolation, ensuring reproducibility of results.

Observing the evolution of results of ANN-benchmarks throughout the last couple of years, and the experimental evaluations of the paper reviewed in this survey, we formulate in the following a few concrete suggestions on how to improve benchmarking of future works.

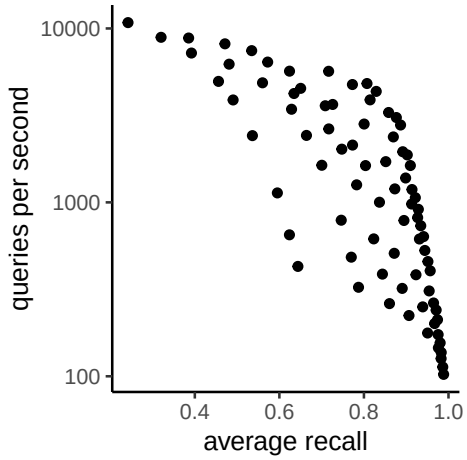


Figure 3: Recall/qps performance of several parameter combinations of HNSW on the Glove dataset.

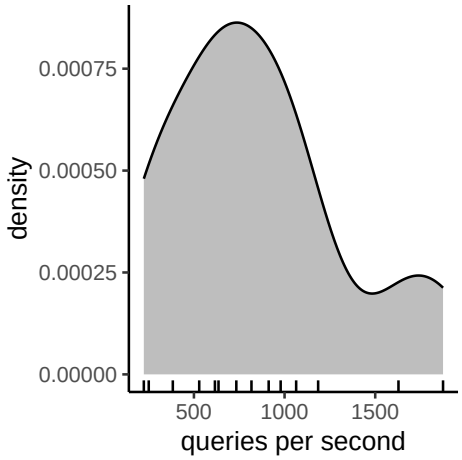


Figure 4: Distribution of queries per second of several parameter combinations of HNSW achieving at least recall 0.9.

3.1 Reporting on multiple metrics

Most of the times, the metrics considered for approximate nearest neighbor queries are the query throughput and the average recall. The query throughput—or equivalently the query time—are however very dependent on the implementation, and are thus measuring the performance of the implementation, rather than assessing the merits of the underlying algorithm [38].

While the actual performance of an implementation is what people are most interested in, considering other metrics can give further insights into the behavior of an algorithm. For instance, the number of distance computations performed can indicate how effective an approach is at reducing the amount of work to be performed. The interplay between the number of distance computations and the actual running time is also of interest: a linear scan through compressed points using product quantization may be faster than a more sophisticated index due to better use of the cache, despite carrying out more distance computations. As highlighted in [19], this is for example true for clustering-based approaches compared to graph-based approaches in million-scale settings. While the latter only carry out a fraction of distance computations, the more efficient memory layout of clustering-based approaches can make up for the additional calculations.

Furthermore, the index size and the index construction time are important metrics to complement the execution speed.

3.2 Selecting parameters

Many papers evaluate the proposed method by comparing with a few state of the art approaches, using a few configurations for each. Some papers use just a single configuration for the baseline, namely the default one provided by the implementation or the ones discussed in the associated publication. This approach can however lead to misleading comparisons, in that the performance of many approaches varies wildly in response to parameter changes, and differently across datasets.

For instance, HNSW is a commonly used baseline to compare with, and in many cases only a few parameter combinations are tested. Figure 3 reports the results—in terms of average recall and queries per second—of

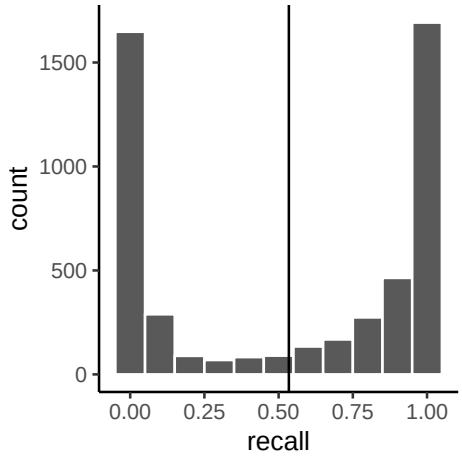


Figure 5: Histogram of the recall of single queries for a configuration of HNSW achieving average recall 0.55.

answering 5 000 queries on `glove-100-angular`[1] using HNSW in the implementation provided by the FAISS library [35]. The index building parameters being used are $M \in \{4, 8, 12, 16, 24, 36, 48, 64, 96\}$ and $efConstruction = 500$; the search parameters are varied in the range $ef \in \{10, 20, 40, 80, 120, 200, 400, 600, 800\}$. As one can observe, the outcomes vary wildly, both in terms of recall and in terms of query throughput. As such, using only a single parameter configuration as a baseline for comparison is very likely to result in suboptimal performance for the baseline itself. In this case we consider HNSW due to its popularity, but the same behavior can be observed with most approaches.

Even fixing a target recall and focusing on a single configuration achieving it does not make the performance more stable across parameter configurations. Figure 4 shows the distribution of the query throughput for all the aforementioned configurations that result in an average recall between 0.9 and 0.95. As we can see, the difference between the slowest and the fastest configurations is about two orders of magnitude.

3.3 Making workloads explicit

Many papers describe which datasets are part of their experimental section and then make a generic statement along the lines of n queries are run from the dataset. The reader is then left to conjecture that possibly queries are sampled at random from the dataset. However, it has been shown [8] that the practical performance of queries is greatly influenced by the intrinsic dimensionality of the queries themselves. Some works already address explicitly workloads with different difficulties [22, 68].

We therefore suggest to explicitly design query workloads that span a different range of difficulties, thus allowing to assess the performance of the approaches under test in finer detail. We provide Python code to compute intrinsic dimensionality measures of workloads at the following public repository: `https://github.com/Cecca/workloads-difficulty`.

3.4 Dangers of reporting on averages

In the previous section we considered the average recall of 5 000 queries as a performance indicator. This can be misleading at times, and hide interesting behavior.

[1] `http://ann-benchmarks.com/glove-100-angular.hdf5`

Consider again HNSW. Figure 5 reports the histogram of the recalls of individual queries of a run attaining average recall ≈ 0.55[2]. Strikingly, almost a third of the queries have recall 0, and a third of the queries have recall 1. In this case considering the average hides this bimodal behavior that may have important practical implications.

Therefore, we suggest to consider the distribution of the performance of individual queries, rather than drawing conclusions based on averages alone.

3.5 Implementation accessibility

The possibility to access the implementation backing the findings reported in any paper is of paramount importance for the community to verify and build upon results. Fortunately, in recent years the number of papers in nearest neighbor search that make their code accessible (usually as a Git repository hosted on online services such as GitHub or BitBucket) increased significantly.

Still, we note that code accessibility can be improved further. In some case, the code linked to a paper fails to compile following the instructions, most often due to differences between the environments of the code's author and the reader. Among the many solutions to this problem, we believe the most straightforward is to pair the code with container environments such as Docker [14] or Singularity [39]. Doing so also makes for an easier integration in existing benchmarking efforts, which often leverage containers in their infrastructure [10].

4 New approaches to ANN search

Having covered general approaches to high-dimensional indexing and remarks on benchmarking efforts, we will now focus on recent work. In the context of this overview paper, we report on approaches that appeared after the publication of the benchmarking paper [10].

4.1 Hashing-based Approaches

Recent works based on hashing have focused on extending classic LSH techniques by using new data structures, new query procedures, and by incorporating information from the data and query distribution.

PUFFINN [9] is an approach whose goal is to address approximate k-nearest neighbor queries while providing theoretical guarantees on the failure probability. In order to do so, it leverages the theoretical framework of Locality Sensitive Hashing (LSH) [15, 27]. While providing theoretical guarantees, LSH is known to have many parameters, whose setting is crucial to achieve good performance. To overcome this issue, PUFFINN adopts an adaptive approach based on the LSH forest trie data structure [13].

PM-LSH [66] is an approach focusing on L_p norms. Their key idea is as follows. A dimensionality reduction using the Johnson-Lindenstrauss transform is applied to project each point into a lower-dimensional space. The transformed points are then indexed by means of a PM-tree [55]. Queries are then carried out by performing several range queries on the PM-tree. While this approach is reminiscent of the earlier SRS approach [58], there is a somehow subtle difference. Where SRS runs k-NN queries in the projected space, which may suffer from the inaccuracy introduced by the projection (i.e. the second nearest neighbor in the projected space might not be the best candidate in the original space) PM-LSH runs a sequence of range queries, which they demonstrate to be more accurate.

FARGO [65] focuses on the Maximum Inner Product Search problem (MIPS). In order to apply LSH to the MIPS problem, the paper proposes an asymmetric transformation of data and queries so that all data points have the same norm, while retaining the original inner products with the queries. Then data points are indexed using LSH, and queried using a multi-probing approach.

[2] Using the `faiss` implementation of HNSW, with $efConstruction = 500$, $M = 16$ and $ef = 10$. Queries are the ones provided in the `glove-100-angular.hdf5` file from ANN-benchmarks.

CEO-MIPS [48] targets the MIPS problem as well, by performing several Gaussian random projections of the data and the queries. By leveraging the theory of concomitants of extreme order statistics, CEO-MIPS considers among all the projections of the query only the one with maximum value. If the i-th projection maximizes the absolute inner product, then it considers as candidates the data points whose i-th projection is large. The paper describes several variants of the approach that reduce the space usage.

FALCONN++ [49] improves on the Cross-Polytope-based hashing used in the FALCONN library [3]. The main insight is that mapping a data point into a bucket based on the random vector that maximizes the inner product (crosspolytope hashing), can also be used as an estimator of the distance between the point and the query vector, similarly to [48]. The paper proposes a threshold for the inner product to keep a point in a bucket, otherwise it is filtered away. This is the first practical implementation using the locality-sensitive filtering technique proposed by Andoni et al. [4] and by Christiani [17] in the context of approximate nearest neighbor search. We note that Rashtchian et al. [51] used this framework for computing similarity joins for skewed data.

LSH-CO-SUBSTRING [40] seeks to overcome one of the main hurdles of using LSH, that is selecting the number of repetitions. The key idea is that, instead of performing L repetitions of the LSH scheme, each vector is associated with a string of hash values of length m. Then, instead of defining buckets, a query looks for the hash strings with the longest colliding subsequence, allowing wraparounds at the string boundaries. The experimental analysis shows that the approach has an edge over other LSH-based approaches in terms of query time at a given recall, with markedly smaller index sizes.

LSH-APG [64] aspires to blend together LSH and graph-based approaches. In particular, LSH is used to speed up the construction of a graph-based index. At query time, LSH is again used to select a few entry points into the graph; these are then used to handle the search for the best query answers. Experiments show that LSH-APG has a larger index than graph-based methods, but such index can indeed be constructed much faster. Furthermore, the query performance at fixed parameters is shown to be better than other graph-based approaches.

DB-LSH [62] employs a dynamic bucketing scheme, modifying the classic LSH approach, to answer range queries. The traditional LSH approach for the Euclidean distance requires to project points onto a random direction, and then to bucket the projections at indexing time to build the hash codes. In DB-LSH such quantization is deferred to query time, so to be able to center the buckets around the query. In particular, at index time the random projections of the dataset points are indexed in an R*-tree. At query time, the R*-tree is used to answer a sequence of (rectangular) range searches, that are equivalent to dynamically bucketing the hash values around the query.

HD-INDEX [7] answers approximate k-nearest neighbor queries with the aim to use less space than LSH. The core idea is to partition the dataset with a regular grid, which is then traversed using a space-filling curve (such as Hilbert of Z-order). Points are then inserted in a tree-like data structure using their position along the space filling curve as keys. The rationale is that points that are close in a geometric sense are also close along the space filling curve. The experiments reported in the paper show that, compared to baselines the HD-INDEX answers queries with a better Mean Average Precision, in a shorter time.

4.2 Graph-based approaches

Graph-based approaches offer among the largest variety of known methods, see for example the survey paper by Wang et al. [63]. In general, recent works have focused on enabling the use of graph-based approaches on larger-than-memory data, on addressing the issue of indexing time, and on designing pruning/refinement strategies for the graph building process.

DISKANN [56] targets approximate nearest neighbor search in an external memory setting, where the size of the dataset makes it impossible to store the index and the data entirely in memory. Therefore, the main aim is to develop an index that minimizes the number of disk reads per query to amortize the disk latency. To this end, DISKANN refines a random k-regular graph using iterated beam searches from a central graph node, the medoid. It refines the graph in two steps with different pruning values. In difference to HNSW, no hierarchy is employed.

ELPIS [11] tackles one of the main issues of graph-based approaches, which is the index construction time. It does so by first partitioning the datasets using a tree-based data structure, and then by using HNSW to build graphs on the leaves. As such, the graph construction is carried out in parallel on smaller subsets of the data. This leads to a faster index construction and to a smaller index as well. In particular, ELPIS builds its tree by employing dimensionality reduction techniques borrowed from the data series community, observing that a high-dimensional vector can be considered as an instance of a data series.

Dobson et al. [19] give a detailed account on scaling graph-based nearest neighbor search implementations to billion-scale datasets. In particular, they describe parallelization techniques to deal with the potential data dependencies that can occur during the parallel insertion of points.

4.3 Clustering-based approaches

SCANN [25] extends the standard inverted file index based on (hierarchical) k-means clustering, designed for inner product spaces. Their motivation is that ℓ_2-based k-means clustering may favor centroids that do not preserve the ordering under inner product similarity. To mitigate this issue, they propose an anisotropic quantization technique which is more accurate for inner product similarity. A significant ingredient to SCANN's performance is a very efficient product quantization implementation based on SIMD instructions as described by Andre et al. [6]. Sun et al. [57] extend the approach with an efficient hyper-parameter selection technique.

4.4 Learning-based approaches

Algorithms with predictions [37] is a recent trend in the development of algorithms and data structures. The idea is that an oracle, for example a machine learning model, gives predictions for data that is stored in the data structure. An overview over progress in this field in general is given by Mitzenmacher and Vassilvitskii [45]. A thorough survey of deep learning-based methods for approximate nearest neighbor search is given in [42].

In the context of ANN search, two main research directions are (i) using a machine learning model to guide the candidate generation and (ii) using a machine learning model to set adaptive stopping criteria in a traditional data structure. In the former, the indexing part is augmented with a machine learning model; in the latter, the search phase is augmented using such a model.

ANN AS MULTILABEL CLASSIFICATION by Hyvönen et al. [31] proposes to formulate the following multi-label classification problem: Given $S \subseteq \mathcal{X}$ and $x \in \mathcal{X}$, let $y_i = [p_i \text{ is a } k\text{-NN of } x \text{ in } S]$. Thus, the (high-dimensional) label $y = (y_1, \ldots, y_n)$ represents the set of k nearest neighbors of x. Given these pairs $\{(x^{(i)}, y^{(i)})\}_{1 \leq i \leq n}$, we can train a classifier for this multi-label classification problem. Applied to space partitioning nearest neighbor search algorithms, such as trees, LSH, or clustering-based variants, the authors show that the pre-dominant approach that collects the points that fall into the same part of the partition (i.e., a leaf or a bucket) is not the natural classifier for the multi-label classification problem. Instead, one should use a majority vote based on groundtruth labels y of these points to decide on the candidates to check.

NEURALLSH [20] proposes to build the k-NN graph on the dataset S, and find a balanced partition of this graph into m disjoint parts. These m parts form the m buckets of the data structure. The label of a point x is an m-bit string, where the i-th bit is set if x has a k-NN in part i of the graph. The labeling is learned by means of a neural network, and the search is guided by predicting bucket probabilities using the neural network, and checking all buckets in sorted order, thresholding at a certain value.

BLISS [26] applies iterative repartitioning by learning the bucket assignment and redistributing points according to the learned assignments, in rounds. More precisely, data points are split up at random into B groups/buckets. The label of a data point x is a length-B bit string; label $i \in \{1, \ldots, B\}$ is set if the nearest neighbor of x is in group i. After learning the assignment, a prediction step is carried out for each data point, the top-K buckets with the highest probability are retrieved, and the data point is moved to the least loaded

bucket among these top-K. The process is repeated R times independently. Experimentally, a small value of R is sufficient, and B is set to roughly $\sqrt{n}$.

LEARNING TO HASH, ROBUSTLY [2] proposes a learned LSH function for binary data under Hamming distance. In particular, rather than sampling bit-coordinates uniformly at random as in the standard LSH scheme, the paper describes a method to optimize the probability distribution over coordinates, for the given dataset. In contrast to the approaches mentioned above, they can guarantee worst-case running time comparable to the best known theoretical approaches, while being able to adapt to the difficulty of a query dynamically.

Li et al. [41] note that most approaches using indexes to reduce the number of candidates to evaluate do not adapt to the difficulty of a query, i.e., use the same stopping condition for all queries. The consequence is that to achieve a good recall a conservative stopping condition needs to be used, thus hurting the performance of easier queries. Therefore, they develop a prediction pipeline based on gradient boosting decision trees, allowing the implementation of an adaptive stopping condition. Such an adaptive stopping condition is then implemented in the IVF and HNSW indexes, with experiments showing a general reduction in latency.

Continuing along the line of the previous paper, Zheng et al. [67] focus on IVF indexes, with the aim of setting the number of cells to probe on a per query basis. To this end, they first modify the way data vectors are clustered, so to ensure that each cluster has a balanced number of entries. Then they use autoencoders, trained offline on sample queries, to estimate the number of cells to probe for a query.

4.5 Refining Candidates

In the context of nearest neighbor search, an important ingredient of the system pipeline is the refinement of a set of candidates. To this end, one wants to compress vectors in a way such that points likely to not be part of the k nearest neighbors are to be excluded without incurring an exact distance computation. A general overview of the techniques is given by Pagh in [46]. The main technique used is Product Quantization as introduced by Jegou et al. [34]. An overview over this technique and its variants is given by Matsui et al. [44].

FINGER [16] aims at improving the speed at which nearest neighbor graphs are traversed. The key observation is that during traversals most of the distances do not need to be computed exactly. Therefore, the paper proposes an estimation method to quickly estimate distances that can be used to improve the performance of any graph-based nearest neighbor algorithm. To showcase the performance of the approach, the paper integrates it in an HNSW implementation.

ADSAMPLING [23] aims at optimizing one of the most basic operations in nearest neighbor search: the distance comparison operation, which given a pair of points returns whether the points' distance is above or below a given threshold. First, the same random rotation is applied to each point. Then, given two vectors their distance is computed by considering the rotated dimensions one at a time, in a progressive sampling fashion, conducting a statistical hypothesis test on whether the two vectors are closer or farther than the threshold.

LVQ [1] describes a locally-adaptive vector quantization technique, which uses scalar quantization with individual lower and upper bounds on the coordinate values. Employed in a graph-based index, the authors show compelling performance up to billion-scale datasets.

5 Trends

5.1 Billion-Scale ANN search

Scaling ANN search to billion-scale datasets has been one of the core research directions of the past years. In particular, this is supported by the availability of diverse datasets through the NeurIPS 2021 Billion-Scale ANN challenge [53] and other large datasets such as Laion5B [52]. On this scale, index construction time and index size pose the most challenging part of the indexing pipeline. Dobson et al. [19] provide an empirical comparison of the scaling of different approaches from million to billion scale, focusing primarily on graph-based approaches.

They discuss different parallelization strategies to efficiently parallelize the index construction and the (batched) search. They conclude that graph-based nearest neighbor search performs best on billion-scale datasets in the high recall regime. In terms of index building times and index size, clustering- and hashing-based approaches were shown to be competitive.

5.2 New Tasks in ANN Search

In the following we describe extension of nearest neighbor search targeted towards real-world applications.

Filtered Search When data vectors are associated with some metadata, for example for the YFCC-100M [61] dataset or the LAION5B dataset [52], a natural extension to k-NN search is to incorporate the metadata into the query. Technically, each data vector is associated with a bag-of-words representation of its tags. The query vector additionally has tags $t_1, \ldots, t_T$, and the task is to find the (approximate) k-NN among all points in the dataset that contain the query's tags. A graph-based solution to this problem is described by Gollapudi et al. [24].

Out-of-Distribution Queries A common scenario in modern approximate nearest neighbor search is that data and query vectors originate from different distributions, but are embedded into the same space. For example, the Yandex TEXT2IMAGE dataset [53] consists of image embeddings produced by the Se-ResNext-101 model described by Hu et al. [28], while queries are textual embeddings produced by a variant of the DSSM model by Huang et al. [29]. As described in [53], the mapping to the shared 200-dimensional real valued vectors under inner product similarity is learned via minimizing a variant of the triplet loss using clickthrough data. A novel approach to explicitly deal with out-of-distribution queries is described in [33]. As pointed out in [19], cluster- and LSH-based approaches are particularly affected by these changes in distribution.

Streaming Search A particular challenge in the index building process is to maintain an index over dynamic insertions and deletions. Technically, a stream of add, delete, search operations is given, where each search is supposed to return the approximate k-nearest neighbors of all the elements that are in the index according to the add and delete operations. Simhadri [3] describes different applications of this setting. For example, for web search the index may contain roughly a trillion vectors and has to handle billions of updates per day. Searches have to be handled at a latency of at most 10 milliseconds with a throughput of 10,000-100,000 queries per second. Naïve solutions such as placing tombstones for deleted items quickly degrade performance, as would a complete rebuild of the index in a given interval size. Singh et al. [54] describe a graph-based approach to handle this issue and show competitive performance to the non-stream setting, and a big improvement over previous LSH-based approaches [59]. Their system handles thousands of add/delete operations per second while maintaining high recall/throughput comparable to the non-streaming setting. A special case of this dynamic setting is content drift, which was studied by Baranchuk et al. [12].

5.3 Conclusions

In recent years, the field of approximate nearest neighbor search has witnessed an exciting surge in the development of new approaches and the refinement of existing methods. Alongside the traditional k-nearest neighbor setting, new tasks are emerging: queries can incorporate different metadata such as tags, an index has to be kept over a dynamic stream of search, insert, and remove operations, or queries might arise from a different distribution than the data points. Moreover, the scale of today's datasets has reached billions of data points, requiring unprecedented scalability while retaining accuracy.

[3] `https://harsha-simhadri.org/pubs/ANNS-talk-Sep22.pptx`, (accessed on Sept 25, 2023.)

Given the practical relevance of the problem, it is crucial that new approaches are benchmarked thoroughly. Given the effort needed to set up a benchmarking infrastructure, we invite the research community to adopt and improve shared benchmarks which avoid the pitfalls that commonly affect experimental evaluations.

Acknowledgements: Martin Aumüller thanks Yusuke Matsui for insightful discussions. He also thanks his collaborators organizing and working on approximate nearest neighbor search challenges [53, 60], and the NeurIPS'23 challenge[4]. This research was supported by the Innovation Fund Denmark for the project DIREC (9142-00001B).

References

[1] Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore Willke. Similarity search in the blink of an eye with compressed indices. Proc. VLDB Endow., 16(11):3433–3446, 2023. ISSN 2150-8097. doi: 10.14778/3611479.3611537. URL https://doi.org/10.14778/3611479.3611537.

[2] Alexandr Andoni and Daniel Beaglehole. Learning to hash robustly, guaranteed. In ICML, volume 162 of Proceedings of Machine Learning Research, pages 599–618. PMLR, 2022.

[3] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In NIPS, pages 1225–1233, 2015.

[4] Alexandr Andoni, Thijs Laarhoven, Ilya P. Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In SODA, pages 47–66. SIAM, 2017.

[5] Alexandr Andoni, Aleksandar Nikolov, Ilya P. Razenshteyn, and Erik Waingarten. Approximate nearest neighbors beyond space partitions. In SODA, pages 1171–1190. SIAM, 2021.

[6] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Quicker ADC : Unlocking the hidden potential of product quantization with SIMD. IEEE Trans. Pattern Anal. Mach. Intell., 43(5):1666–1677, 2021.

[7] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. Proc. VLDB Endow., 11(8): 906–919, 2018.

[8] Martin Aumüller and Matteo Ceccarello. The role of local dimensionality measures in benchmarking nearest neighbor search. Inf. Syst., 101:101807, 2021.

[9] Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Michael Vesterli. PUFFINN: parameterless and universally fast finding of nearest neighbors. In ESA, volume 144 of LIPIcs, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[10] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. Inf. Syst., 87, 2020.

[11] Ilias Azizi, Karima Echihabi, and Themis Palpanas. Elpis: Graph-based similarity search for scalable data science. Proc. VLDB Endow., 16(6):1548–1559, 2023.

[4] https://big-ann-benchmarks.com/

[12] Dmitry Baranchuk, Matthijs Douze, Yash Upadhyay, and I. Zeki Yalniz. Dedrift: Robust similarity search under content drift. CoRR, abs/2308.02752, 2023.

[13] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In WWW, pages 651–660. ACM, 2005.

[14] Carl Boettiger. An introduction to docker for reproducible research. ACM SIGOPS Oper. Syst. Rev., 49(1): 71–79, 2015.

[15] Andrei Z. Broder. On the resemblance and containment of documents. In SEQUENCES, pages 21–29. IEEE, 1997.

[16] Patrick H. Chen, Wei-Cheng Chang, Jyun-Yu Jiang, Hsiang -Fu Yu, Inderjit S. Dhillon, and Cho-Jui Hsieh. FINGER: fast inference for graph-based approximate nearest neighbor search. In WWW, pages 3225–3235. ACM, 2023.

[17] Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In SODA, pages 31–46. SIAM, 2017.

[18] Tobias Christiani. Fast locality-sensitive hashing frameworks for approximate near neighbor search. In SISAP, volume 11807 of Lecture Notes in Computer Science, pages 3–17. Springer, 2019.

[19] Magdalen Dobson, Zheqi Shen, Guy E. Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. Scaling graph-based ANNS algorithms to billion-size datasets: A comparative analysis. CoRR, abs/2305.04359, 2023.

[20] Yihe Dong, Piotr Indyk, Ilya P. Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In ICLR. OpenReview.net, 2020.

[21] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. Proc. VLDB Endow., 13(3): 403–420, 2019. doi: 10.14778/3368289.3368303. URL `http://www.vldb.org/pvldb/vol13/p403-echihabi.pdf`.

[22] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Hercules against data series similarity search. Proc. VLDB Endow., 15(10):2005–2018, 2022.

[23] Jianyang Gao and Cheng Long. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. Proc. ACM Manag. Data, 1(2):137:1–137:27, 2023.

[24] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. Filtered-DiskANN: Graph algorithms for approximate nearest neighbor search with filters. In WWW, pages 3406–3416. ACM, 2023.

[25] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In ICML, volume 119 of Proceedings of Machine Learning Research, pages 3887–3896. PMLR, 2020.

[26] Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J. Smola. BLISS: A billion scale index using iterative re-partitioning. In KDD, pages 486–495. ACM, 2022.

[27] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. Theory Comput., 8(1):321–350, 2012.

[28] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In CVPR, pages 7132–7141. Computer Vision Foundation / IEEE Computer Society, 2018.

[29] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. Learning deep structured semantic models for web search using clickthrough data. In CIKM, pages 2333–2338. ACM, 2013.

[30] Ville Hyvönen, Teemu Pitkänen, Sotiris K. Tasoulis, Elias Jaasaari, Risto Tuomainen, Liang Wang, Jukka Corander, and Teemu Roos. Fast nearest neighbor search through sparse random projections and voting. In IEEE BigData, pages 881–888. IEEE Computer Society, 2016.

[31] Ville Hyvönen, Elias Jääsaari, and Teemu Roos. A multilabel classification framework for approximate nearest neighbor search. In NeurIPS, 2022.

[32] Elias Jääsaari, Ville Hyvönen, and Teemu Roos. Efficient autotuning of hyperparameters in approximate nearest neighbor search. In PAKDD (2), volume 11440 of Lecture Notes in Computer Science, pages 590–602. Springer, 2019.

[33] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. OOD-DiskANN: Efficient and scalable graph ANNS for out-of-distribution queries. CoRR, abs/2211.12850, 2022.

[34] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell., 33(1):117–128, 2011.

[35] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. IEEE Trans. Big Data, 7(3):535–547, 2021.

[36] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In ICLR. OpenReview.net, 2020.

[37] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In SIGMOD Conference, pages 489–504. ACM, 2018.

[38] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? Knowl. Inf. Syst., 52(2):341–378, 2017.

[39] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. PLOS ONE, 12(5):1–20, 05 2017. doi: 10.1371/journal.pone.0177459. URL https://doi.org/10.1371/journal.pone.0177459.

[40] Yifan Lei, Qiang Huang, Mohan S. Kankanhalli, and Anthony K. H. Tung. Locality-sensitive hashing scheme based on longest circular co-substring. In SIGMOD Conference, pages 2589–2599. ACM, 2020.

[41] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. Improving approximate nearest neighbor search through learned adaptive early termination. In SIGMOD Conference, pages 2539–2554. ACM, 2020.

[42] Mingjie Li, Yuan-Gen Wang, Peng Zhang, Hanpin Wang, Lisheng Fan, Enxia Li, and Wei Wang. Deep learning for approximate nearest neighbour search: A survey and future directions. IEEE Trans. Knowl. Data Eng., 35(9):8997–9018, 2023.

[43] Yury A. Malkov and Dmitry 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.

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

[45] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. Commun. ACM, 65(7): 33–35, 2022.

[46] Rasmus Pagh. Similarity sketching. In Encyclopedia of Big Data Technologies. Springer, 2019.

[47] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In EMNLP, pages 1532–1543. ACL, 2014.

[48] Ninh Pham. Simple yet efficient algorithms for maximum inner product search via extreme order statistics. In KDD, pages 1339–1347. ACM, 2021.

[49] Ninh Pham and Tao Liu. Falconn++: A locality-sensitive filtering approach for approximate nearest neighbor search. In NeurIPS, 2022.

[50] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In ICML, volume 139 of Proceedings of Machine Learning Research, pages 8748–8763. PMLR, 2021.

[51] Cyrus Rashtchian, Aneesh Sharma, and David P. Woodruff. Lsf-join: Locality sensitive filtering for distributed all-pairs set similarity under skew. In WWW, pages 2998–3004. ACM / IW3C2, 2020.

[52] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5B: an open large-scale dataset for training next generation image-text models. In NeurIPS, 2022.

[53] Harsha Vardhan Simhadri, George Williams, Martin Aumü ller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. In NeurIPS (Competition and Demos), volume 176 of Proceedings of Machine Learning Research, pages 177–189. PMLR, 2021.

[54] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. FreshDiskANN: A fast and accurate graph-based ANN index for streaming similarity search. CoRR, abs/2105.09613, 2021.

[55] Tomás Skopal, Jaroslav Pokorný, and Václav Snásel. Nearest neighbours search using the pm-tree. In DASFAA, volume 3453 of Lecture Notes in Computer Science, pages 803–815. Springer, 2005.

[56] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. Rand-nsg: Fast accurate billion-point nearest neighbor search on a single node. In NeurIPS, pages 13748–13758, 2019.

[57] Philip Sun, Ruiqi Guo, and Sanjiv Kumar. Automating nearest neighbor search configuration with constrained optimization. In ICLR. OpenReview.net, 2023.

[58] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. Proc. VLDB Endow., 8(1):1–12, 2014.

[59] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. Proc. VLDB Endow., 6(14):1930–1941, 2013.

[60] Eric S. Tellez, Martin Aumüller, and Edgar Chavez. Overview of the SISAP 2023 indexing challenge. In SISAP, 2023.

[61] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. YFCC100M: the new data in multimedia research. Commun. ACM, 59(2):64–73, 2016.

[62] Yao Tian, Xi Zhao, and Xiaofang Zhou. DB-LSH: locality-sensitive hashing with query-based dynamic bucketing. In ICDE, pages 2250–2262. IEEE, 2022.

[63] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. Proc. VLDB Endow., 14(11):1964–1978, 2021.

[64] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. Proc. VLDB Endow., 16(8):1979–1991, 2023.

[65] Xi Zhao, Bolong Zheng, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. FARGO: fast maximum inner product search via global multi-probing. Proc. VLDB Endow., 16(5): 1100–1112, 2023.

[66] Bolong Zheng, Xi Zhao, Lianggui Weng, Quoc Viet Hung Nguyen, Hang Liu, and Christian S. Jensen. PM-LSH: a fast and accurate in-memory framework for high-dimensional approximate NN and closest pair search. VLDB J., 31(6):1339–1363, 2022.

[67] Bolong Zheng, Ziyang Yue, Qi Hu, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. Learned probing cardinality estimation for high-dimensional approximate NN search. In ICDE, pages 3209–3221. IEEE, 2023.

[68] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. Generating data series query workloads. VLDB J., 27(6):823–846, 2018.

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314

Non-profit Org.
U.S. Postage
PAID
Los Alamitos, CA
Permit 1398