Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art

ILIAS AZIZI, UM6P, Université Paris Cité, Morocco - France
KARIMA ECHIHABI, UM6P, Morocco
THEMIS PALPANAS, Université Paris Cité, France

Vector data is prevalent across business and scientific applications, and its popularity is growing with the proliferation of learned embeddings. Vector data collections often reach billions of vectors with thousands of dimensions, thus, increasing the complexity of their analysis. Vector search is the backbone of many critical analytical tasks, and graph-based methods have become the best choice for analytical tasks that do not require guarantees on the quality of the answers. We briefly survey in-memory graph-based vector search, outline the chronology of the different methods and classify them according to five main design paradigms: seed selection, incremental insertion, neighborhood propagation, neighborhood diversification, and divide-and-conquer. We conduct an exhaustive experimental evaluation of twelve state-of-the-art methods on seven real data collections, with sizes up to 1 billion vectors. We share key insights about the strengths and limitations of these methods; e.g., the best approaches are typically based on incremental insertion and neighborhood diversification, and the choice of the base graph can hurt scalability. Finally, we discuss open research directions, such as the importance of devising more sophisticated data-adaptive seed selection and diversification strategies.

CCS Concepts: • **Theory of computation** → **Graph algorithms analysis**; **Nearest neighbor algorithms**; **Data structures design and analysis**; • **Information systems** → **Information retrieval**; • **General and reference** → *Evaluation*; *Experimentation*.

Additional Key Words and Phrases: Vector similarity search, Approximate nearest neighbor, KNN graph analysis, Seed selection, Neighborhood diversification, Graph algorithms

ACM Reference Format:
Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 43 (February 2025), 31 pages. https://doi.org/10.1145/3709693

1 Introduction

Vector data is common in various scientific and business domains, and its prevalence is expected to grow in the future with the proliferation of learned embeddings [43, 107]. The volume and dimensionality of this data, which can exceed multiple terabytes and thousands of dimensions, make its analysis very challenging [107]. A critical component of these data analysis tasks is vector search [40, 42, 43, 107]. It supports recommendation [91, 139], information retrieval [154], clustering [21, 150], classification [114] and outlier detection [19, 20, 22, 78, 118] in many fields including bioinformatics, computer vision, security, finance and medicine. In data integration, vector search

Authors' Contact Information: Ilias Azizi, UM6P, Université Paris Cité, Morocco - France, ilias.azizi@um6p.ma; Karima Echihabi, UM6P, Morocco, karima.echihabi@um6p.ma; Themis Palpanas, Université Paris Cité, France, themis@mi.parisdescartes.fr.

ACM 2836-6573/2025/2-ART43
https://doi.org/10.1145/3709693

is used for entity resolution [38], missing values imputation [61], and data discovery [164]. Besides, it is exploited in software engineering [4, 105] to automate API mappings and in cybersecurity to profile network usage and detect malware [37]. More recently, vector search has been playing a crucial role in improving the performance and interpretability of Large Language Models and reducing their hallucinations [18, 76, 83, 84].

A vector search algorithm over a dataset S of n d-dimensional vectors returns answers in S that are similar to a given input vector V_Q. The brute-force approach (a.k.a. a sequential or serial scan) compares V_Q to every single element in S. The time complexity of this approach is $O(nd)$, which becomes impractical when dealing with large collections of data with high dimensionality. State-of-the-art approaches typically improve this time complexity by reducing the dimensionality d using concise and precise summarization techniques and/or decreasing the number of processed vectors n, by developing efficient indexing data structures and search algorithms that prune unnecessary comparisons to V_Q. Some approaches compute exact answers, while others ϵ- and δ-ϵ-approximate answers (with deterministic and probabilistic guarantees, respectively, on the accuracy of the answers), or ng-approximate answers (without any theoretical guarantees, but high accuracy in practice) [44, 45]. One popular vector search algorithm is k-nearest neighbor (k-NN) search, which returns the k vectors in S closest to V_Q according to a similarity measure, such as the Euclidean distance.

The efficiency of exact vector search has significantly improved over the last decade [25, 44, 108, 112, 148]. However, exact techniques still do not address the query latency constraints of many analytical tasks. Therefore, a large body of work has been dedicated to approximate vector search, which trades off accuracy for efficiency [45, 86, 147]. These approaches are based on scans [49, 152], trees [6, 23, 26, 81, 89, 101, 110, 113, 143, 144, 146, 149, 156], graphs [52, 54, 97, 102, 129, 140], inverted indexes [12, 70, 71, 155], hashing [67, 131, 132], or a combination of these data structures [11, 29, 52, 72, 102, 153, 163]. Over the last decade, graph-based techniques have emerged as the method of choice for vector search in many real applications that can relax theoretical guarantees to achieve a query latency of a few milliseconds on terabyte-scale collections [73, 125, 128, 162].

Figure 1 illustrates vector search in an image retrieval use case. We produce embeddings for ImageNet [121] using a ResNet50 model [65]. We report the time at which a best-so-far (bsf) answer is found (i.e., the image in the database most similar to the query) by vector search techniques from different families: ELPIS [11] and EFANNA [52] for ng-approximate search, QALSH [66] for δ-ϵ-approximate search, and a serial scan for exact search. We observe that: (i) ELPIS returns the same answer as the serial scan and QALSH over three orders of magnitude faster, which explains the popularity of graph-based vector search in many real applications; and (ii) not all graph-based methods have the same performance, e.g. ELPIS is 3x faster than EFANNA, which motivates this experimental study to help the community better understand the strengths and limitations of the existing graph-based vector search techniques.

Graph-based methods typically structure a dataset S as a proximity graph $G(V, E)$ where V is the set of n vertices representing the n data points in S, and E is the set of edges that connect similar vertices. Answering a query V_Q usually consists of running an ng-approximate beam search algorithm that warms up the results queue with an initial list of candidate nodes in G known as seed nodes, selects one of them as an entry node, i.e., the node that will be traversed first, and performs a greedy graph traversal following the edges of the traversed nodes to return all nodes that are similar to V_Q. State-of-the-Art (SotA) graph-based vector search methods, such as HNSW [97], NSG [54], ELPIS [11], Vamana [129], and NSSG [53] adopt the same beam search algorithm to retrieve ng-approximate answers from the graph. However, they differ in how they construct the graph and initiate the beam search.

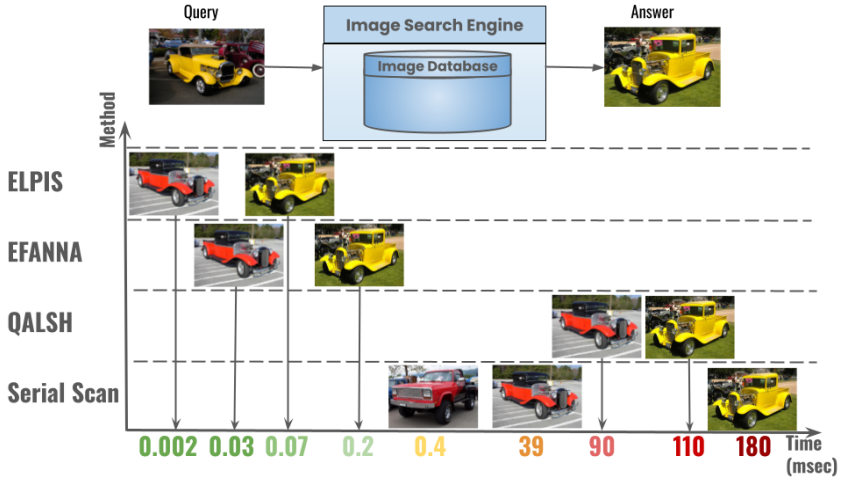


Fig. 1. Image retrieval using different vector search methods. Each row shows the bsf answer returned by each method (y-axis) at a given timestamp (x-axis). The graph-based approach ELPIS: (i) returns the same answer as the serial scan and QALSH over three orders of magnitude faster; and (ii) is 3x faster than the graph-based method EFANNA.

A number of studies have evaluated vector search methods. Some focus on exact search [44] while others cover approximate search with and without guarantees [7, 85, 103]. Given the increased interest in graph-based vector search, a recent experimental evaluation [142] has been dedicated entirely to this family of techniques. However, the results of this study are inconclusive, because the experimental evaluation was conducted on very small datasets, not exceeding 1M vectors (we will show in Section 4 that trends change as dataset size increases, e.g. some methods that are efficient on 1M datasets do not perform well on large collections and vice-versa). Besides, our study proposes a new taxonomy that classifies SotA graph-based vector search methods according to five key design paradigms. It also highlights the impact of these design choices on the indexing and search performance, and how this impact changes with dataset size. In parallel, some benchmarks [8, 126] also evaluate vector search methods, but they do not focus on graph-based approaches, and thus, do not shed light on why the best methods have superior query performance. In contrast, our study covers real datasets reaching one billion vectors from different domains such as deep network embeddings, computer vision, neuroscience, and seismology, includes more recent approaches, and uncovers interesting insights that have not been reported before, such as the impact of seed selection and neighborhood diversification on indexing and search performance.

In this paper, we make the following contributions.

• We identify five main paradigms exploited by state-of-the-art vector search methods: (1) Seed Selection (SS) determines the node(s) in the graph where the search initiates. (2) Neighborhood Propagation (NP) approximates the k-NN graph, where each node is connected to its exact k nearest neighbors, by propagating neighborhood lists between connected nodes. (3) Incremental Insertion (II) builds a proximity graph incrementally connecting vertices with short links to their closest neighbors and long links that model small-world phenomena. (4) Neighborhood Diversification (ND) sparsifies the graph by pruning edges that lead to redundant directions at each node. (5) Divide-and-Conquer (DC) splits a large dataset into partitions and builds individual graphs separately. We provide a brief overview of these five paradigms, and study in depth the two key

paradigms that have a strong impact on query performance (as shown in Section 4), namely SS and ND.
• We propose a new taxonomy that categorizes graph-based vector search methods according to the five paradigms described earlier, and highlights the chronological development of the different techniques and their influence map.
• We provide a brief survey of the SotA graph-based vector search methods. We describe their design principles and pinpoint their strengths and limitations.
• We conduct an exhaustive experimental evaluation of twelve SotA methods using synthetic and real-world datasets from neuroscience, computer vision and seismology, with sizes reaching up to 1 billion vectors. We provide a rich discussion of the results. We confirm some conventional wisdom, e.g., methods that construct graphs through incremental insertion exhibit superior query performance and scalability compared to other existing methods [11, 142]. We also explain results that differ from the literature, in particular with respect to the most recent study on the same topic [142]. For instance, in the recent study, SPTAG-BKT [29] and HNSW [97] rank low in terms of query and indexing efficiency, respectively. However, our experiments show that SPTAG-BKT is among the most efficient techniques in query answering on small-sized datasets (1M-25GB), and that many methods lag behind HNSW in terms of indexing time, including on small datasets. In addition, contrary to previous works [11], we demonstrate that Vamana [129] is also competitive with ELPIS and HNSW [11, 97].
• We share new insights about vector search that have never been published before. For instance, (i) we show that not all ND techniques are equally effective, and we identify the one that is most effective for large dataset sizes; and (ii) we illustrate how different SS strategies affect query answering and index building (some methods perform a beam search during graph construction).
• Based on the results of the comprehensive experimental study, we pinpoint some promising research directions in this field: (i) novel graph structures that can scale to large data collections should be proposed for methods based on NP and ND strategies; (ii) more effective and data-adaptive seed selection strategies should be developed to further improve indexing and search performance on large-scale datasets; (iii) a deeper theoretical understanding of ND approaches could pave the way for the development of enhanced methods that can adapt to data distributions; and (iv) devising techniques (e.g., clustering, neighborhood diversification) tailored to DC-based methods has the potential to improve their performance.
Scope: The goal of this study is to conduct an experimental evaluation of the SotA graph-based methods designed for in-memory dense vector search. A theoretical analysis of the indexing and search algorithms is an open research question (no existing graph-based vector search method provides this analysis), and is out-of-scope. The extension to the out-of-core scenario and the theoretical analysis are both part of our future work.

2 Preliminaries

The ng-approximate vector search problem is typically modeled as an ng-approximate k-NN search problem in high-dimensional vector space. Data points are represented as d-dimensional vectors in R^d, and the *dissimilarity* between the points is measured using the Euclidean distance *dist*. We consider a dataset collection $S = \{V_{C_1}, V_{C_2}, \dots, V_{C_n}\}$ of n d-dimensional points and a query vector V_Q.

Definition 1. *Given a positive integer k, an **exact k-NN query** over a dataset S retrieves the set of vectors $A = \{\{V_{C_1}, \dots, V_{C_k}\} \subseteq S \mid \forall V_C \in A$ and $\forall V_{C'} \notin A$, $dist(V_Q, V_C) \leq dist(V_Q, V_{C'})\}$ [45].*

Definition 2. *(An equivalent definition has appeared in [45].) Given a positive integer k, an **ng-approximate k-NN query** over a dataset S retrieves the set $A = \{V_{C_1}, ..., V_{C_k}\} \subseteq S$ in a heuristic manner, i.e., there are no theoretical guarantees about the quality of the answers.*

We use the terms vector search, similarity search, ng-approximate search, and approximate search interchangeably.

2.1 Approximate Vector Search Methods

Vector search has been heavily studied for over fifty years. Proposed approaches are either based on scans, trees, graphs, hashing, inverted indexes, or on hybrid designs combining two or more of these data structures. Vector search methods often rely on summarization to reduce the complexity of the search. In this section, we provide an overview of the main summarization techniques exploited by state-of-the-art approaches, then describe the fundamental concepts of each family, discussing its strengths and limitations.

Summarization Techniques *Random projections* project the original high-dimensional vector into a lower dimensional space using a random matrix. The pairwise distances are guaranteed to be nearly preserved with high probability if the dimension of the projected space is large enough [74]. *Quantization* compresses a vector of infinite values to a finite set of codewords that constitute the codebook. *Scalar* quantization maps each vector dimension independently, whereas *vector* quantization maps the whole vector at once exploiting inter-dimension correlations [60]. *Product* quantization [70] divides the vector and operates a vector quantizer on the subvectors. *Optimized Product Quantization* (OPQ) optimizes the space decomposition of a product quantizer by decorrelating the dimensions. The *Karhunen-Loève Transform (KLT)* [75, 92] performs a linear transformation that adapts to a given signal to decorrelate the dimensions, then applies scalar quantization. Recent work has shown that some summarizations that were developed for data series are equally efficient for generic vectors [41, 45]. In particular, *Piecewise Aggregate Approximation* (PAA) [77] and *Adaptive Piecewise Constant Approximation* (APCA) [24] segment a vector into equal or variable length segments, respectively, and summarize each segment with its mean value. *Extended APCA* (EAPCA) [146] improves upon APCA by using both the mean and standard deviation to summarize each segment. *Symbolic Aggregate Approximation* (SAX) [87] transforms a vector using PAA and discretizes the values using a scalar quantizer. *Deep Embedding Approximation* (DEA) learns summarizations using the SEAnet deep neural network [145].

Tree-based indexes organize the data using a tree structure and have been the method of choice for exact vector search for data series and generic high-dimensional vectors [16, 25, 27, 28, 39, 48, 62, 82, 90, 111–113, 122, 145, 146, 148, 157, 166]. Some works have proposed using tree indexes to support approximate search with [30, 45] or without guarantees [6, 45, 101, 148]. The tree-based methods that support ng-approximate search either build multiple randomized K-D Trees that are searched in parallel [101], segment the space into smaller dimensions indexed by an RDB tree [6], or use heuristics to select candidates from some leaf node(s) [45, 148]. The ng-approximate search accuracy/efficiency tradeoff is tuned with user-defined parameters specific to each method, e.g., specifying the maximum number of leaves to visit during search.

Scan-based approaches have been designed primarily for exact search. Sequential scans compare the query to each single candidate in a dataset. Skip-sequential scans summarize the high-dimensional data in a filter file that fits in memory and only search the original data, typically stored out-of-core, if a candidate cannot be pruned using the filter file. Some scan-based methods can also support ANN search with [45] or without guarantees [45, 49, 152]. These methods tune the ng-approximate search accuracy/efficiency tradeoff with a user-defined parameter that specifies the maximum number of vectors to be visited.

Inverted indexes typically compress the original high-dimensional vector data using quantization to reduce the space overhead and improve efficiency. Popular methods such as the Inverted Multi-Index (IMI) [12], and the IVF-PQ [70] organize the data by associating lists of data points (a.k.a. posting lists) with their proximity to a codeword. These codewords are representative points, e.g. the medoid/centroid of the posting list, derived after clustering the original data, collectively forming a codebook. The ng-approximate search algorithm accesses the codebook to retrieve points linked to the closest codeword(s). The accuracy/efficiency tradeoff is tuned with a user-defined parameter that specifies the maximum number of posting lists to be explored during the search. This tradeoff is also affected by the index building parameters such as the codebook size and the quantization bit budget.

Hash-based approaches belong primarily to the family of locality-sensitive hashing (LSH) [67, 69]. They are designed to support δ-ϵ-approximate vector search [45] by exploiting hash functions that group with high-probability similar data points into the same bucket and non-similar data points into different buckets. Numerous LSH variants exist, each with its unique strengths. Among these, the SRS [132] and QALSH [66] methods stand out. SRS provides efficient search results with a relatively small index size, while QALSH enhances bucketing precision by using the incoming query as a reference point. The δ-ϵ-approximate search accuracy/efficiency tradeoff is tuned with user-defined parameters that specify the approximation error and probability thresholds. This tradeoff is also affected by the index building parameters, e.g., the number of hash tables and the number of random projections.

Graph-Based approaches support ng-approximate vector search. They often employ a proximity graph structure [56, 123, 137], where each vertex represents a data point, and edges connect similar vertices. Two vertices are linked if the data points they represent are close in their respective spaces, usually determined by a common distance measure such as the Euclidean distance [46]. When searching for a specific query point, the process typically begins from a set of initial points or seeds. These seeds can be chosen randomly or based on certain criteria. The search uses one of these seeds as an entry node and the others as initial candidate answers. It then progresses by visiting the neighboring vertices of the current node in a best-first, greedy manner, concluding when no better matches are identified [119]. State-of-the-art graph-based methods, including KGraph [35], EFANNA [52], HNSW [97], DPG [86], NSG [54], SPTAG [141], SSG [53], Vamana [129], HCNNG [102], ELPIS [11] and others [63, 72, 96] use the same search algorithm [119] but differ in how they construct the graph and select seed points.

Hybrid approaches combine ideas from the different families described earlier. IEH [72] retrieves initial neighbors to build the graph using hash-based methods, such as LSH [57] and ITQ [161]. EFANNA [52] builds an initial graph using randomized K-D Trees where nodes are connected to the neighbors returned by K-D Trees, and exploits these trees during search to find initial candidates. SPTAG [29] divides the dataset using multiple Trinary-Projection Trees (TP Trees) [141] before merging multiple graphs constructed for each of the subsets, and constructs multiple K-D Trees or BKTrees to retrieve initial seeds. HCNNG [102] divides the dataset using random hierarchical clusterings, and builds multiple Minimum Spanning Tree (MST) structures on the different subsets. The graph is built by merging the multiple MSTs. HCNNG also exploits K-D Trees to retrieve seeds to initiate the search. ELPIS [11] builds a Hercules tree [41] to divide the dataset into subsets, and constructs an HNSW [97] graph on each subset. During search, ELPIS prunes subsets using the EAPCA [146] lower-bounding distance.

Summary. Tree-based approaches typically demonstrate efficient indexing performance, boasting low index construction time and memory usage [11, 45, 141]. Many of these methods support various search flavors, from exact to approximate, with some offering guarantees. However, tree-based methods often fall short in search efficiency, especially when faced with challenging query

Algorithm 1: Beam Search (G, V_Q, s, k, L)

Input: Graph G, query vector V_Q, initial seeds s, result size k, beam width $L \geq k$
Output: k approximate nearest neighbors to V_Q

initialize candidate set $C \longleftarrow s$;
initialize visited list $R \longleftarrow \emptyset$;
while $C \backslash R \neq \emptyset$ **do**
  let $p^* \longleftarrow \text{argmin}_{V_i \in C \backslash R}\, dist\,(V_Q, V_i)$;
  update $C \longleftarrow C \cup N_{out}\,(p^*)$;
  update $R \longleftarrow R \cup p^*$;
  if $|C| > L$ **then**
    update C to retain closest L points to V_Q;
return the k candidates in C closest to V_Q;

workloads [45]. Methods based on inverted indexes are less scalable in terms of indexing time and footprint than tree-based indexes but can answer queries faster. However, achieving high query accuracy typically requires a significant indexing time and space overhead [45]. Hash-based methods provide theoretical guarantees on query accuracy and are the only SotA approximate methods that support theoretical guarantees on query performance. However, building the index requires a significant overhead both in space and time, often requiring different indexes to be pre-built to support different guarantees during search. Graph-based approaches, while expensive in terms of indexing time and memory usage, and lacking guarantees on query acccuracy, showcase impressive empirical query accuracy and efficiency [11, 45, 86, 88].

3 Graph-Based Vector Search

We now present an overview of the main SotA graph-based ng-approximate vector search methods. We outline the base data structures and algorithms in this field, and identify five main paradigms exploited by the SotA approaches. We propose a new taxonomy that categorizes these approaches along the five paradigms, highlighting also their chronological development and influence map.

3.1 A Primer

A proximity graph is a graph $G\,(V, E)$ in which two vertices V_i and V_j are connected by an edge if and only if they satisfy particular geometric requirements, namely the *neighborhood criterion* [124]. A proximity graph can be constructed using different distances such as the dot product [100], nevertheless the Euclidean distance remains the most popular one [46]. One of the earliest proximity graphs in the literature is the Delaunay Graph (*DG*). It is a planar dual graph for the Voronoi Diagram [51], where each vertex is the center of its own voronoi cell, and two vertices are linked if and only if their corresponding voronoi cells share at least one edge. A DG satisfies the Delaunay Triangulation :$\forall q, p, r \in V, (q, p)\,, (q, r)\,, (r, p) \in E$ if the circumcircle of the triangle q, p, r is empty [10]. A beam search [119] (Algorithm 1) on a *DG* can find the exact nearest neighbors [34] when the dataset has a high dimensionality or the beam search uses a large beam width. However, using a DG in high dimensions is impractical, because the graph becomes almost fully connected as the dimensionality d grows [34]. Thus, SotA methods build alternative graph structures and use beam search to support efficient query answering [56, 99, 137].

3.2 Main Paradigms

We provide a brief overview of the five main paradigms exploited by SotA methods. Then, we describe in more detail the two paradigms that have the greatest impact on query performance (as will be demonstrated in Section 4).

Seed Selection (SS) chooses initial nodes to visit during search. It is also used during index building by approaches that exploit a beam search during the construction of the index to decide which edges to build. Some methods simply select one or more seed(s) randomly, while others use special data structures, e.g., a K-D Tree.

Neighborhood Propagation (NP) refines a pre-existing graph following a user-defined number of iterations, a.k.a. NNDescent [36]. During each iteration, the potential neighbors of a given node are sourced both from its immediate neighbors and the neighbors of its neighbors. Then, the node only keeps the m closest neighbors, where m is a user-parameter. The pre-existing graph could be a random graph or some other type of graph.

Incremental Insertion (II) refers to building a graph by inserting one vertex at a time. Each vertex is connected using bi-directional edges to its nearest neighbors and some distant vertices. The neighbors are selected using a beam search on the already inserted portion of the graph. At the end of graph construction, some vertices retain early connections which act as long-range links. This approach was first proposed in the object-based peer-to-peer overlay network VoroNet [14], with the idea of adding long-range links being inspired from Kleinberg's small-world model [79, 80], with the difference that the latter selects the long-range links randomly.

Neighborhood Diversification (ND) was first introduced by the Relative Neighborhood Graph (RNG) [136]. It aims to sparsify the graph by pruning unnecessary edges while preserving connectivity. For each node, ND exploits the geometrical properties of the graph to remove edges to neighbors that lead to redundant regions or directions This indirectly causes the creation of long-range links allowing nodes to maintain diversified neighborhood lists, which reduces the number of comparisons during search.

Divide-and-Conquer (DC) is a strategy that splits a dataset into multiple, possibly overlapping, partitions, then builds a separate graph on each partition. Some approaches such as SPTAG [29] and HCNNG [102] combine the individual graphs into one large graph, on which a beam search is performed, while ELPIS [11] maintains the graphs separate and searches them in parallel.

3.3 Seed Selection

While SotA graph-based vector search methods adopt diverse strategies for constructing the graph, they virtually all use beam search for query answering (Algorithm 1). because it usually retrieves good answers if the graph is well-connected. However, choosing the right nodes to visit first has an impact on how quickly good answers are found. The longer the graph traversal, the higher the number of visited nodes, and the slower the search. Several methods build one or more index(es), in addition to the graph, on top of a sample of data points. These additional indexes are used during query answering to find the entry points in the graph. However, to the best of our knowledge, none of these methods have provided sufficient theoretical or empirical evidence to support their choices for seed selection. In this paper, we conduct an in-depth study of the different seed selection techniques proposed in the literature:

(1) **Stacked-NSW (SN)**: Inspired by skip lists [116], HNSW [97] constructs hierarchical multi-resolution graphs [96] for seed selection. Each level contains a diversified NSW graph built using a sample of nodes from the level below, with the lowest layer sampling from the entire dataset. Stacked NSW is constructed by assigning each node a maximum level L. Nodes with L>0 are

incrementally inserted into the graphs from layer L down to 1. The maximum level L is:

$$L = \frac{-\ln(\xi)}{\ln(M \div 2)} \qquad \text{(Eq. 1 [97])}$$

where ξ is a uniformly random number between 0 and 1, and M is the maximum out-degree in the graphs, controlling the probability distribution of a node's maximum layer. A high M decreases the number of nodes represented in hierarchical layers as well as the number of layers, while a lower M allocates more nodes to the hierarchical layers, thus generating more hierarchical layers. During query answering, it starts a greedy search from a fixed entry point at the top layer, descending layer by layer, each time starting from the closest node found in the previous layer, until reaching the bottom layer. The node selected in the bottom layer and the nodes connected to it serve as seed nodes.

(2) **K-D Trees (KD)**: Utilized by EFANNA [52], SPTAG-KDT [140], and HCNNG [102], this technique involves constructing single or multiple K-D Tree(s) [17] on a dataset sample. During search, a set of seed points is retrieved by running a depth-first search traversal (DFS) on the K-D Tree structure(s), warming up the set of candidate answers. The node closest to the query is selected as an entry node.

(3) **LSH**: Utilized by IEH [72], this strategy constructs an LSH index on a sample of the dataset and uses it during search to return a set of seed points, using one as an entry node.

(4) **Medoid (MD)**: This strategy fixes the medoid node as entry point during query answering and uses its neighbors as seeds.

(5) **Single Fixed Random Entry Point (SF)**: A random node is selected and fixed as the entry point for all searches. This node and the nodes connected to it are used as seeds.

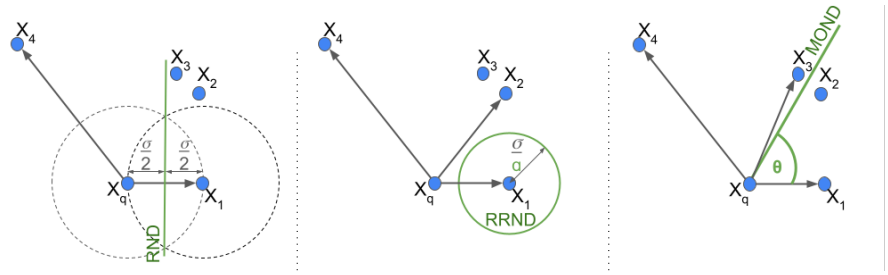
(6) **K-Sampled Random Seeds (KS)**: For each query, k random nodes are selected to warm up the set of candidate answers. This approach is supported in DPG [86], NSG [54], and Vamana [129] which choose the medoid as the entry point and enhance the list of initial seeds with the random nodes.

(7) **Balanced K-means Trees (KM)**: Utilized by SPTAG-BKT [140], this technique constructs Balanced K-means Trees (BKT) [95] on a dataset sample. During search, seed points are retrieved via depth-first search (DFS) on the BKT structure(s).

3.4 Neighborhood Diversification

The goal of Neighborhood Diversification (ND) is to create a sparse graph structure, where nodes are connected both to their close neighbors and to some further nodes. This is because traversing a graph in which nodes are only connected to their close neighbors incurs many unnecessary comparisons before reaching the promising regions. This method first appeared in the Relative Neighborhood Graph (RNG) [136, 137], which builds an undirected graph from scratch or by pruning an existing Delaunay Graph, then removes the longest edge in each triangle of three connected points. This approach, and other variations that exploit different geometric properties of the graph, have been adapted for directed graphs by several graph-based ng-approximate vector search methods [11, 29, 53, 54, 86, 97, 129]. We identify three main ND strategies used by these methods: Relative Neighborhood Diversification (RND), Relaxed Relative Neighborhood Diversification (RRND) and Maximum-Oriented Neighborhood Diversification (MOND). Note that the ND strategy is different from Kleinberg's small world network graph [79, 80]. The long-range links in the latter are achieved through a random selection of nodes, while in ND they result indirectly from pruning the close neighbors of each node. Assume:

- X_q is the query node, i.e. the node to be inserted in the graph.
- R_q, the list of current closest neighbors to X_q.



(a) RND **(b) RRND** **(c) MOND**

Fig. 2. Neighborhood diversification approaches

- C_q, the list of candidate neighbors to X_q not yet in R_q.
- X_j is a candidate from C_q considered for inclusion in R_q.
- X_i is a node already in the set R_q.
- dist (X_i, X_j) is the Euclidean distance between X_i and X_j.

Definition 3 (RND). *The node X_j is added to R_q if and only if the following condition holds:*

$$\forall X_i \in R_q,\ dist\left(X_q, X_j\right) < dist\left(X_i, X_j\right) \quad (Eq.\ 2)$$

Definition 4 (RRND). *For a relaxation factor $\alpha \geq 1$, the node X_j is added to R_q if and only if the following condition holds:*

$$\forall X_i \in R_q,\ dist\left(X_q, X_j\right) < \alpha \cdot dist\left(X_i, X_j\right) \quad (Eq.\ 3)$$

Definition 5 (MOND). *For an angle $\theta \geq 60°$, the node X_j is added to R_q if and only if the following condition holds:*

$$\forall X_i \in R_q,\ \angle\left(X_j X_q X_i\right) > \theta \quad (Eq.\ 4)$$

Figure 2 shows the results after applying different ND approaches on the candidate neighborhood list $C_q = \{X_1, X_2, X_3, X_4\}$ for node X_q. RND (Figure 2a) was first used in the context of vector search by HNSW [97]. Other methods adopted RND including NSG [54], SPTAG [29] and ELPIS [11]. Per Definition 3, RND operates over the candidate neighborhood list by adding X_j to R_q if its distance to X_q is smaller than the distance between X_j and any neighbors X_i of X_q. In the example of Figure 2a, X_q is going to connect to X_1 since it is the closest neighbor ($\sigma = dist\left(X_q, X_1\right)$). Both X_2 and X_3 are pruned from the X_q neighborhood list as they are closer to X_1 than X_q, and X_4 will be added to X_q eventually since dist(X_q,X_4) $\leq$ dist(X_1,X_4). In contrast, RRND (Figure 2b), proposed by Vamana [129], introduces a relaxation factor α (with $\alpha \geq 1.5$) where X_j is added to R_q if its distance to X_q is smaller than α times its distance to X_i, a neighbor of X_q in R_q, relaxing the property to prune less candidate neighbors (hence X_2 will not be pruned in this case, but X_3 is pruned due to its proximity to X_2). When $\alpha = 1$, RRND is reduced to RND. Finally, MOND (Figure 2c) was proposed by DPG [86] and used in NSSG [53] as well. It aims at maximizing angles in the graph topology (cf. Definition 5), guiding the selection process via the angle threshold θ (e.g., $\theta = 60°$). MOND prunes the candidate neighbors of a node to favor edges pointing in different directions (X_2 is pruned since $\angle X_1 X_q X_2 < 60°$, while X_q connects with X_3 since $\angle X_1 X_q X_3 > 60°$ and with X_4 since $\angle X_1 X_q X_4 < 60°$ and $\angle X_3 X_q X_4 < 60°$). Note that any nodes pruned by RRND and MOND will eventually be pruned by RND, but not vice versa. Refer to [3] for a detailed proof.

3.5 A Taxonomy

Figure 3 depicts the SotA graph-based approaches, classified based on the five design paradigms: SS, NP, II, ND, and DC. The taxonomy also reflects the chronological development of the methods. Directed arrows indicate the influence of one method on another. Within the ND category, distinctions are made between different strategies, i.e., No Neighborhood Diversification (NoND), RND, RRND, and MOND (cf. Section 3.4). We identify the SS strategy of each method: KS, KD, SN, MD, LSH, and KM (SF is not used by any SotA method, but we consider it as an alternative strategy). Additionally, some methods use more than one strategy (e.g. NSG and VAMANA use KS and MD), or offer the flexibility to use different strategies (e.g., SPTAG can use either KD or KM). Note that a method can exploit one or more paradigms; e.g., HNSW uses incremental node insertion and prunes each node's neighbors using the RND approach, thereby being classified as both II and ND. KGraph [35] was the first to use NP to approximate the exact k-NN graph (k-NNG) (with quadratic complexity), and influenced numerous subsequent methods, including IEH [72] and EFANNA [52]. In parallel, NSW [115] introduced the II strategy for graph construction. HNSW [97] and DPG [86] leveraged ND to enhance NSW and KGraph [35], respectively. The good performance of HNSW and DPG encouraged more methods to adopt the ND paradigm, including NGT [158], NSG [54] and SSG [53], which apply ND on the NP-based graph EFANNA [52]. SPTAG [29] combined DC with ND. Vamana [129] adopts NSG's idea of constructing the graph through beam search and ND. However, Vamana constructs its graph by refining an initial base random graph in two rounds of pruning, using RRND and RND. Inspired by HNSW, Vamana and NGT proposed variants that support incremental graph building [31, 158], but we classify them as ND-based per the ideas proposed in the original papers. HCNNG [102] was influenced by SPTAG [29] and adopted a DC approach for constructing the graph without adopting ND. ELPIS [11] also adopted a DC strategy but leveraged both II and ND. HVS [94] and LSHAPG [163] both propose new seed selection structures for HNSW, with the latter additionally adopting a new probabilistic rooting approach. Note that earlier approaches, except from NSW, were mainly NP-based; however, recent studies have focused on devising methods that leverage the ND, II, and DC paradigms because they lead to superior performance (cf. Section 4).

3.6 State-of-the-Art Approaches

KGraph [35] reduces the construction cost of an exact k-NNG, which has a quadratic worst-case complexity. It constructs an approximate k-NNG by refining a random initial graph with an empirical cost of $O\left(n^{1.14}\right)$ [36]. This refinement process, also known as NNDescent [36] (Neighborhood Propagation), aims at improving the approximation of the k-NN graph by assuming that the neighbors of a vertex u are more likely to be neighbors of each other. The process iterates over all graph vertices $u \in V$: for each vertex u and pair (x, y) of its neighbors, it adds x to the neighbors of y and vice-versa, keeping the closest k neighbors of u.

Navigable Small World (NSW) [96, 115] is an approximation of a Delaunay graph which guarantees the small world property [151], i.e. the number of hops L between two randomly chosen vertices grows to the logarithm of graph size n such that $L \propto Log\left(n\right)$. An *NSW* graph is based on the VoroNet graph [14], an extention of Kleinberg's variant of Watts-Strogatz's small world model graph [79, 80], The VoroNet graph is built incrementally by inserting a randomly picked vertex to the graph and connecting it to 2d+1 neighbors selected using a beam search on the existing vertices in the graph. Once this process completes, the first built edges would serve as long-range edges to quickly converge toward nearest neighbors [14]. The resulting graph was proved to guarantee the small world network property [14, 15].

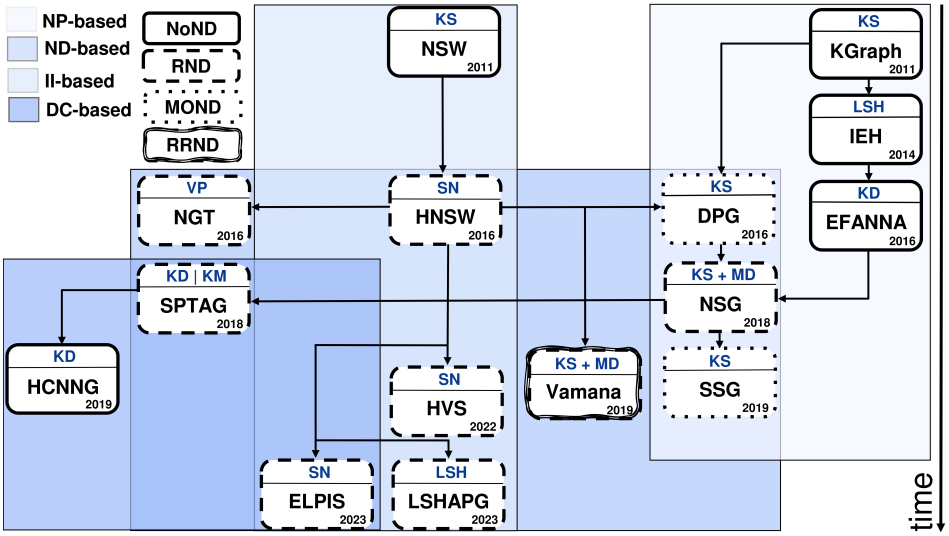


Fig. 3. Graph-based ANN indexing paradigms

Iterative Expanding Hashing (IEH) [72] follows the same process as KGraph to construct an approximate k-NNG; however, it refines an initial graph where the candidates for each node are generated using a hashing function. Two extensions of IEH have been proposed to better leverage advanced hashing methods for generating initial candidates: IEH-LSH [57] and IEH-ITQ [161]. All these methods use NNDescent to finalize the graph connections.

EFANNA [52] selects seeds similarly to KGraph [35] and IEH [72] and refines candidates using NNdescent. It builds an approximate k-NNG by selecting initial neighbors of each node using randomized truncated K-D Trees [32] and refining the graph using NNDescent [36]. During search, EFANNA uses the pre-built trees to select seeds, then runs a beam search on the graph index.

Hierarchical Navigable Small World (HNSW) [97] improves the scalability of NSW [96, 115] by proposing RND to sparsify the graph and a hierarchical seed selection strategy (SN) to shorten the search path during index building and query answering. Each hierarchical layer includes all nodes in the layer above it, with the bottom (a.k.a. *base*) layer containing all points of the dataset S, HNSW builds an NSW graph incrementally. However, HNSW diverges from NSW in that it refines the candidate nearest neighbors, identified through beam search on the nodes already in that layer using RND. During query answering, HNSW utilizes SN to quickly find an entry point in the base layer to start the beam search.

Diversified Proximity Graph (DPG) [86] extends KGraph [35] by diversifying the neighborhoods of its nodes through edge orientation, a technique we refer to as Maximum-Oriented Neighborhood Diversification (MOND) in Section 3.4. MOND's main objective is to maximize the angles between neighboring nodes, contributing to a sparsed graph structure. This process is iteratively applied to all nodes. After that, the directed graph is transformed into an undirected one, enhancing its connectivity. Nevertheless, note that DPG's publicly available implementation [33] utilizes RND rather than MOND for neighborhood diversification.

NGT [158] is an approximate nearest neighbor (ANN) search library developed by Yahoo Japan. It offers two construction methods: one extends KNN graphs with reverse edges, forming bi-directed KNN graphs [68], while the other incrementally builds graphs similar to HNSW with a range-based search strategy [130]. In this study, we consider the former [68]. Additionally, the library includes

methods that employ quantization for highly efficient search. NGT maintains efficiency by pruning neighbors via RND and using Vantage-Point Trees [160] to select seed nodes for accurate query results.

Navigating Spreading-out Graph (NSG) [54], similarly to DPG, builds an approximate k-NNG first. But, unlike DPG, it builds an EFANNA graph rather than a KGraph. It then diversifies the graph using RND. At the end, NSG creates a depth-first search tree to verify the connectivity of the graph. If there is a vertex that is disconnected from the tree, NSG connects it to the nearest node in the tree to ensure graph connectivity.

SPTAG [29] is a library for approximate vector search proposed by Microsoft. SPTAG follows a DC approach and is based on multiple existing works. It selects small dataset samples on which it builds either K-D Trees [17] or Balanced K-means Trees [95]. These strutures will be used for seed selection during query answering. Then it clusters the full dataset using multiple hierarchical random divisions of TP Trees [141], builds an exact k-NN graph on each cluster (i.e., leaf) and refines each graph using ND. The graphs are merged into one large graph index for query processing.

Vamana [129] is similar to NSG in considering the set of visited nodes when building long-range edges within the graph. However, instead of using EFANNA [52], Vamana uses a randomly generated graph with node degree $\geq\ log\,(n)$ to ensure the initial graph connectivity [47]. Then, for each node, Vamana runs a beam search on the graph structure to get the visited node list R, which will be refined in the first round using RRND. After adding bi-directional edges to selected neighbors, the neighbors that exceed the maximum allowed out-degree will refine their neighborhood list following an RND process. Then, Vamana repeats the same refinement process a second time to improve the graph quality, this time using RRND with $\alpha \geq 1$ to increase the connectivity within the graph.

SSG [53] integrates the MOND approach from DPG [86] and closely follows the steps of NSG [54] and DPG [86] in index building from a foundational graph. Instead of performing a search for each node to acquire candidates, SSG [53] employs a breadth-first search on each node to assemble candidate neighbors through local expansion on a base graph (EFANNA). When the maximum size for the candidate neighbors is achieved, SSG reduces the neighbors in the list by enforcing the MOND diversification strategy, pruning the candidate nodes forming an angle smaller than a user-defined parameter θ with the already existing neighbors of the concerned node. After iteratively applying this method to all nodes, SSG [53] enhances connectivity by constructing multiple DFS trees from various random points, in contrast to NSG's [54] singular DFS approach.

Hierarchical Clustering-based Nearest Neighbor Graph (HCNNG) [102] was inspired by SPTAG. It employs hierarchical clustering to randomly divide the dataset into multiple subsets. This subdivision process is executed several times, resulting in a collection of intersecting subsets. On each subset, HCNNG constructs a Minimum Spanning Tree (MST) graph. Following this, the vertices and edges from all the MSTs are merged to form a single, connected graph. To facilitate the search process, HCNNG constructs multiple K-D Trees [17], to identifying entry points during query search.

HVS [94] extends HNSW's base layer by refining the construction of hierarchical layers. Instead of random selection, nodes are assigned to layers based on local density to better capture data distribution. Each layer forms a Voronoi diagram and uses multi-level quantization, increasing dimensionality by a factor of 2 in each lower layer. Search at the base layer is similar to that of HNSW.

LSHAPG [163] combines HNSW graphs with multiple hash tables based on the LSB-Tree structure [133] to enhance search efficiency. It leverages L hash tables to retrieve seeds for beam search on the base layer, unlike HNSW, which selects a single seed through SN. LSHAPG also utilizes

these hash tables for probabilistic rooting during search, pruning neighbors based on the projected distance before evaluating and pruning the raw vectors.

ELPIS [11] is a DC-based approach that splits the dataset into subsets using the Hercules EAPCA tree [41], where each leaf corresponds to a different subset, then builds in parallel a graph-based index for each leaf using HNSW [97]. During search, ELPIS first selects heuristically an initial leaf and executes a beam search on its respective graph. The retrieved set of answers feed the search priority queues for the other leaves. Only a subset of leaves is selected based on the answers and the lower-bounding distances of the query to the EAPCA summarization of each leaf. Then, ELPIS initiates multiple concurrent beam searches on the graph structures of the candidate leaves. Finally, ELPIS aggregates all results from candidate clusters and returns the top-k answers.

4 Experimental Evaluation

We experimentally evaluate twelve state-of-the-art graph-based vector search methods, based on the two key paradigms described in Section 3, i.e., SS and ND. To single out the effect of each strategy, we first implement a basic II-based method, where nodes are inserted incrementally and each node i acquires its list of candidate neighbors through a beam search on the current partial graph of already inserted nodes. Then, we implement each strategy independently on the resulting graph. Finally, we assess the indexing and query-answering performance of these methods on a variety of real and synthetic datasets. All artifacts are available in [3].

4.1 Framework

Setup. Methods were compiled with GCC 8.2.0 under Ubuntu Linux 20.04 (Rocky Linux 8.5 on HPC) using default compilation flags and optimization level 3. Experiments were conducted on an Intel Xeon Platinum 8276 server with 4 sockets, each with 28 cores and 1 thread per core. The CPU cache configuration is: 32KB L1d, 32KB L1i, 1,024KB L2, and 39,424KB L3 cache. The server includes a 1.5TB RAM via 24x 64GiB DDR4 DIMMs.

Algorithms. We cover the following methods: HNSW [97], NSG [54], Vamana [129], DPG [86], EFANNA [52], HCNNG [102], KGraph [35], NGT [158], DPG [86], and two versions of SPTAG [29] (SPTAG-BKT and SPTAG-KDT, using BKT and K-D Trees, respectively). We also include ELPIS [11] and LSHAPG [163], new techniques not evaluated in the latest survey [142]. IEH [72] and FANNG [63] are excluded due to suboptimal performance [54, 142], and HVS [94] due to difficulties running the official implementation [93]. We use the most efficient publicly available C/C++ implementations for each algorithm, leveraging multithreading and SIMD vectorization to optimize performance. We also carefully inspected all code bases and, as is common in the literature [31, 55, 120, 142, 159, 165], disabled the optimizations that would lead to an unfair evaluation such as cache pre-warming in Vamana and L2-normalized Euclidean distance in NSG, EFANNA, and Vamana. Since all methods except ELPIS and HNSW use a single linear buffer as a priority queue, we modified the original implementations of these two algorithms (which used two max-heap priority queues) [1, 2]. The modifications to each code base are documented in [3].

Datasets. We use seven real-world datasets from various domains, including deep network embeddings, computer vision, neuroscience, and seismology: (i) *Deep* [127] contains 1 billion 96-dimensional vectors extracted from the final layers of a convolutional neural network; (ii) *Sift* [71, 135] consists of 1 billion 128-dimensional SIFT vectors representing image feature descriptors; (iii) *SALD* [138] provides neuroscience MRI data with 200 million 128-dimensional data series; (iv) *Seismic* [50] contains 100 million 256-dimensional time series representing earthquake recordings from seismic stations worldwide; (v) *Text-to-Image* [13] offers 1 billion 200-dimensional image embeddings from Se-ResNext-101 along with 50 million DSSM-embedded text queries for cross-modal retrieval under domain shifts; (vi) *GIST* [70] contains 1 million 960-dimensional vectors,

using GIST descriptors [106] to capture spatial structure and color layout of images; and (vii) *ImageNet1M*, a new dataset that we generated from the original ImageNet [121], producing embeddings of 1 million original vectors using the ResNet50 model [65], with PCA applied to reduce dimensionality to 256. We select subsets of different sizes from the Sift, Deep, SALD and Seismic datasets, and we refer to each subset with the name of the dataset followed by the subset size in GBs (e.g., Deep25GB). We refer to the 1-million and 1-billion datasets with the 1M and 1B prefixes, respectively. To evaluate the methods on datasets with different distributions, we generate three random 25GB datasets RandPow0, RandPow5 and RandPow50, each with 256 dimensions, following the power law distribution [104] using three power law exponents: 0 (uniform [117]), 5 and 50 (very skewed). The power law distribution models many real world phenomena (including in economics, physics, social networks, etc.). It is a relationship of type $Y = kX^a$, where Y and X are variables of interest, a is the power law exponent and k is a constant. The skewness of a dataset distribution increases with a. When $a = 0$, the dataset is evenly distributed.

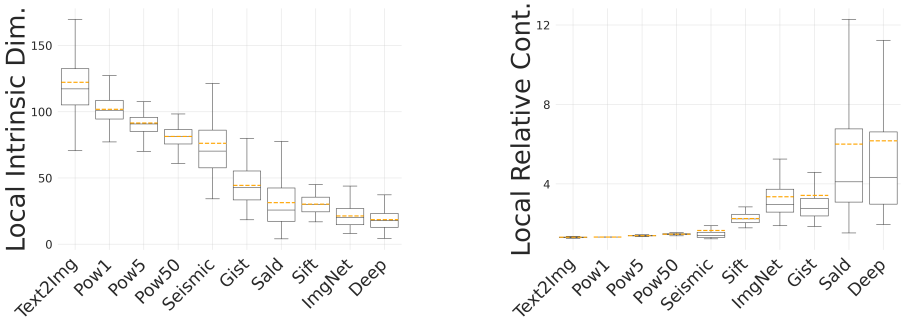
Dataset Complexity. The complexity of the datasets in our experimental evaluation is assessed using Local Intrinsic Dimensionality (LID) [5, 9] and Local Relative Contrast (LRC) [9, 64]. The LID and LRC for a query point x are defined as follows:

$$\text{LID}(x) = -\left(\frac{1}{k}\sum_{i=1}^{k}\log\frac{\text{dist}_i(x)}{\text{dist}_k(x)}\right)^{-1} \qquad \text{(Eq. 5)}$$

$$\text{LRC}(x) = \frac{\text{dist}_{\text{mean}}(x)}{\text{dist}_k(x)} \qquad \text{(Eq. 6)}$$

where $dist_i(x)$ is the Euclidean distance between point x and its i-th true nearest neighbor, k is the number of nearest neighbors, and $\text{dist}_{mean}(x)$ is the average distance of x to all other points in the dataset. LID represents the intrinsic dimensionality of the data (and can be significantly lower than the original dimensionality of the data space): the lower the LID, the easier the search is. LRC is an intuitive measure of separability of the nearest neighbor of a query from the rest of the points in the dataset: the higher the LRC, the easier the search is. Figure 4 shows that the LID and LRC results are consistent. The graphs were produced using a subset of 1M points randomly sampled from each dataset, and $k = 100$. Note that the orange horizontal lines in Figures 4a and 4b denote the mean LID and LRC values of each dataset, respectively. The Pow0, Pow5, Pow50, Seismic and Text2Img datasets have the highest LID and lowest LRC values, indicating that they are hard datasets for vector search [9]. Sift, Deep and ImageNet, on the other hand, are the easiest datasets in our workload, with the lowest LID and highest LRC values.

Queries. Query sets include 100 vectors processed sequentially, not in batches, mimicking a real-world scenario where queries are unpredictable [58, 59, 109]. Results with 1 million queries are extrapolated from 100 query sets. For Deep, Sift, GIST, and Text-to-Image, queries are randomly sampled from available query workloads. For SALD, ImageNet, and Seismic, 100 queries are randomly selected from the datasets and excluded from the index-building phase.For hardness experiments, we use Deep query vectors of varying complexity, denoted as a percentage ranging from 1% to 10%. These vectors were obtained by adding Gaussian noise ($\mu = 0$, σ^2 ranging from 0.01 to 0.1) to randomly choose dataset vectors, with the percentage reflecting the σ^2 value [167]. The 100-query workloads for the power law distribution datasets are generated following the same distributions using different seeds. Unless otherwise stated, all experiments were conducted with 10-NN queries per the standard in the community [2, 8, 126]. We use 100-NN queries to evaluate dataset complexity because a higher k improves the estimation of LID and LRC (Eqs. 5-6), and to assess the different SS strategies because the higher the k the more overhead for seed selection.



(a) Local Intrinsic Dimensionality (LID): low values indicate easy search

(b) Local Relative Contrast (LRC): high values indicate easy search

Fig. 4. Dataset Complexity

Measures. We measure the *wall clock time* and *distance calculations* for both indexing and query answering. We also measure the accuracy of each k-NN query using *Recall* which quantifies the fraction of the true nearest neighbors that the query S_Q successfully returns.
Procedure. We tune each method to achieve the best trade-offs in accuracy/efficiency. Then, we carry experiments in two steps: indexing building and query answering, with caches cleared before each step and kept warm during the same query workload. Methods were allowed at most 48 hours to build a single index. During timed experiments, the server was used exclusively to ensure accurate measurements. For each query workload, we ran the experiment six times; we excluded the two best and worst, and reported the mean of the remaining performances. For reproducibility, all parametrization details are provided in [3].

4.2 Neighborhood Diversification

We now evaluate the ND strategies covered in Section 3, i.e., RND, RRND, and MOND against a baseline without ND (NoND). We apply each strategy individually to an II-based graph, where each node is inserted sequentially and linked with a pruned list of neighbors, determined via a beam search with maximum out-degree $R = 60$ and beam width $L = 800$. Bi-directional edges are added to neighbors, and the neighborhood list is pruned to size R using the same ND strategy. Graphs are built on Deep and Sift (25GB, 100GB and 1B). For RRND and MOND, we run experiments with different values of α $(1 - 2)$ and θ $(50^\circ - 80^\circ)$, respectively, and use $\alpha = 1.3$ and $\theta = 60^\circ$ because they lead to the best performance. Then, we execute workloads with 100 queries against each dataset, and measure the accuracy/efficiency tradeoff using the recall and the number of distance calculations incurred during the search. The results in Figure 5 indicate that both RND and MOND consistently outperform, followed by RRND. NoND is the worst performer overall. As the dataset size increases, the performance gap between NoND and ND methods widens, particularly at high Recall (Figures 5f, 5c). This is due to the higher number of hops needed to find the answers and the density of the neighborhoods in the NoND nodes since no pruning was applied. These results indicate the key role played by the ND paradigm in improving query-answering performance and the superiority of the RND and MOND strategies.

In Table 1, we report the pruning ratios of the three neighborhood ND methods on the Deep and Sift 25GB datasets. The pruning ratio quantifies the percentage reduction in the size of the candidate neighbor list before and after the diversification step. Higher pruning ratios indicate more aggressive pruning, which directly affects the graph size and memory usage. RND achieves

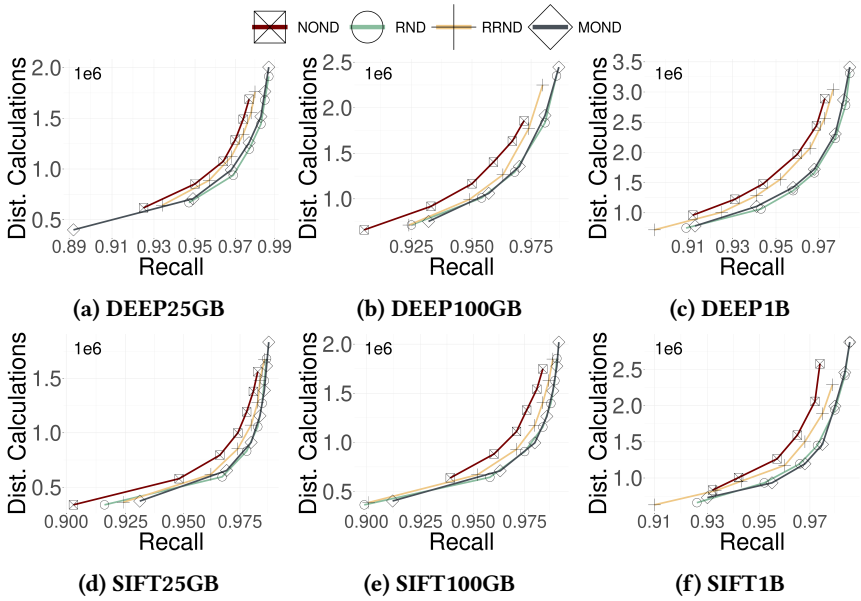


Fig. 5. ND methods performance on real-world datasets

	RND	MOND	RRND
Deep	20%	2%	0.6%
Sift	25%	4%	0.7%

Table 1. Pruning ratios of ND methods on Deep and Sift datasets.

the highest pruning ratios, MOND provides moderate pruning, and RRND exhibits the least pruning. As a result, RND leads to smaller graph sizes and reduced memory requirements, while RRND creates larger graphs with higher memory usage.

4.3 Seed Selection

In these experiments, we focus on the four most common SS strategies for the beam search algorithm: SN [11, 97], MD [54, 129], KS [35, 53, 86, 115, 129], and KD [29, 52, 102] (KM and LSH were excluded because they are not among the commonly used seed selection strategies in graph-based methods). We consider the baseline method SF which has not been used in the literature before. In these experiments, we focus on the four most common SS strategies for the beam search algorithm: SN [11, 97], MD [54, 129], KS [35, 53, 86, 115, 129], and KD [29, 52, 102] (KM and LSH were excluded because they are not among the commonly used seed selection strategies in graph-based methods). We consider the baseline method SF which has not been used in the literature before. These strategies are compared using the same insertion-based graph structure that exploits RND pruning since this is the best baseline from the results in Section 4.2. We run 100 queries for each strategy on the Deep and Sift datasets with sizes 25GB, 100GB, and 1B. We extrapolate the results to 1M queries and report the number of distance calculations to achieve a 0.99 accuracy in Figure 6. We observe that SN and KS are the most efficient strategies across all scenarios, while SF and MD are the least efficient overall. The KD strategy is competitive on 25GB and 100GB Deep and Sift datasets but

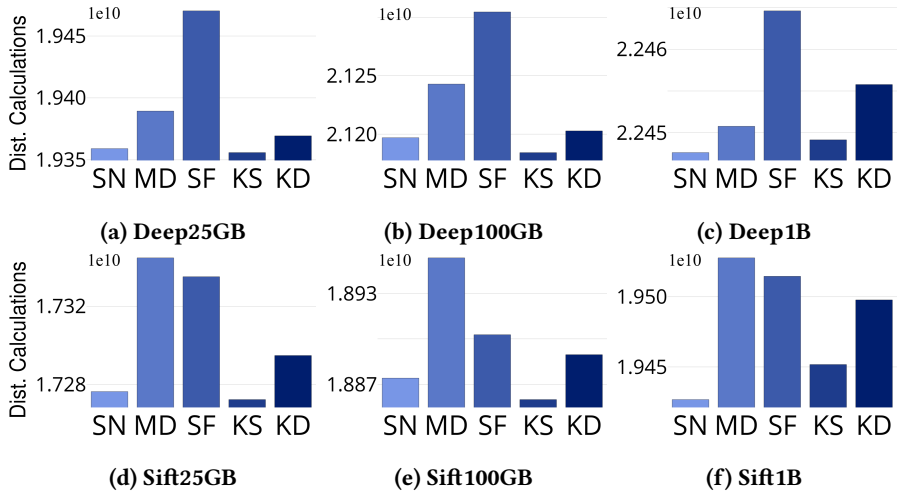


Fig. 6. The impact of SS Methods on query answering

its performance deteriorates on billion-scale datasets. KS outperforms SN on dataset sizes 25GB and 100GB; however, this trend reverses with the 1B size dataset. The difference in distance calculations between SN and KS on the 25GB and 1B datasets is ∼1M and ∼10M, respectively. As the dataset size increases, it becomes imperative to sample more nodes (beyond the beam width utilized during search in KS) to obtain a representative sample of the dataset, thereby enhancing the likelihood of initiating the search closer to the graph region where the query resides (SN adapts its size logarithmically with the growth of the dataset, leading to a better representation of the dataset). Figure 6 also illustrates that both MD and SF are among the least performing strategies, with MD performing better than SF on Deep and vice-versa on Sift. This indicates neither MD nor SF are effective and robust seed selection strategies.

We now study the effect of SS strategies on indexing performance. We focus on the two best strategies KS and SN and study their effect on the same baseline based on II and RND [11, 29, 53, 54, 86, 97, 115, 129]. This is because these methods are the most impacted by the SS strategy used, since they perform a beam search, which includes a seed selection step, at the insertion of each node. We build an index using each strategy on Deep1M and Deep25GB and measure distance calculations. We calculate the distance overhead of SN compared to KS, and we estimate the number of additional 100-NN queries that the KS-based graph can answer, with 0.99 recall, before the SN-based graph completes its construction. We observe (Table 2) that the SN-based graph requires 182 million and 22.3 billion more distance calculations than the KS-based graph on Deep1M and Deep25GB respectively. Furthermore, the KS-based graph can answer 45K and 1.17 million queries on Deep1M and Deep25GB respectively before the SN-based graph finishes construction.

4.4 Indexing Performance

We evaluate twelve state-of-the-art vector search methods, varying dataset sizes and reporting total indexing time and memory footprint. For brevity, we present results only for the Deep dataset as trends are consistent across other datasets. Full results are in [3]. We use subsets of Deep ranging from 1 million to 1 billion vectors (equivalent to 350GB). Indexes are built to allow a 0.99 recall efficiently. Initial experiments on 1 million vectors include all methods. Methods that could not

	Deep1M	Deep25GB
Dist. Calculations (SN)	4.3 billion	1.49 trillion
Dist. Calculations (KS)	4.1 billion	1.46 trillion
Overhead (SN vs. KS)	182 million	22.3 billion
Additional Queries	44,959	1,165,870

Table 2. The impact of SS methods on Indexing Performance

scale to larger datasets are excluded from subsequent experiments. Specifically, HCNNG, SPTAG-BKT, NGT, and SPTAG-KDT take over 24 hours to build indexes on 25GB datasets and exceed 48 hours on 100GB datasets. KGraph, DPG, EFANNA, and LSHAPG delivered unsatisfactory results on 25GB, so they were not included in larger datasets.Furthermore, KGraph and EFANNA require over 300GB and 1.4TB of RAM for 25GB and 100GB datasets, respectively. As DPG, NSG, and SSG rely on KGraph and EFANNA, they were also excluded from larger datasets.

Indexing Time. Figure 7 demonstrates that II-based approaches have the lowest indexing time across dataset sizes. In particular, the II and DC-based approach ELPIS, is 2.7x faster than HNSW and 4x faster than NSG for both 1M and 25GB dataset sizes, while HNSW is 1.4x faster than NGT. Note that NSG's indexing time includes both the construction of its base graph, EFANNA, which is time-intensive, and the refinement with NSG. SPTAG-BKT and SPTAG-KDT exhibit high indexing times, requiring over 25 hours to index the Deep25GB dataset-24 times more than ELPIS, the fastest method. This inefficiency in SPTAG arises from its design, which involves constructing multiple TP Trees and graphs, becoming increasingly costly with larger datasets. On datasets with 100GB and 1B vectors (≈350GB), only HNSW, ELPIS, and Vamana scale with acceptable indexing time, with ELPIS being 2 and 2.7 times faster than HNSW and Vamana, respectively.

Indexing Footprint. Figure 8 reports the memory footprint for each index, including the raw data. To perform the evaluation, we record the peak virtual memory usage during construction[1]. SPTAG-BKT and SPTAG-KDT demonstrate efficient memory utilization (1M and 25GB) despite having the highest indexing time. For larger datasets, ELPIS has the lowest indexing memory footprint, occupying up to 40% less memory than HNSW and 30% less than Vamana during indexing. This is because ELPIS needs a smaller maximum out-degree and beam width compared to its competitors. HNSW has a higher indexing memory footprint due to its use of a graph layout optimized for direct access to node edges through a large contiguous block allocation [1]. This layout offers a time advantage over adjacency lists by reducing memory indirections and cache misses. However, it can result in quadratic memory growth when using a large maximum out-degree on large-scale datasets. In Figure 9, we compare the size of method indices, including the raw data. The figure shows that certain methods, such as EFANNA, HCNNG, KGraph, and consequently NSG, SSG, and DPG (which use one of these base graphs), exhibit a significantly larger memory footprint relative to their final index size. For instance, HCNNG consumes substantial memory during indexing, requiring over 200GB for Deep25GB (Fig. 8) due to merging multiple MST from numerous samples generated during hierarchical clusterings. In contrast, its final index size is less than 50GB (Fig. 9).

4.5 Search Performance

We now evaluate the search performance of the different methods. All methods were included in the 1M experiments. Some methods were excluded in 25GB plots (KGraph, DPG, SPTAG-KDT,

[1]Readings are taken from the proc pseudo filesystem's Virtual Memory Peak.

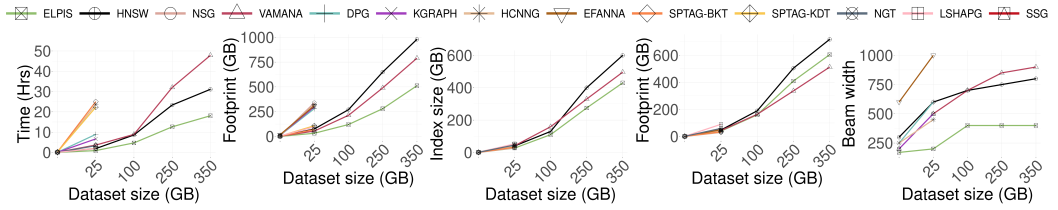


Fig. 7. Indexing Time

Fig. 8. Indexing Memory Footprint

Fig. 9. Indexing Disk Footprint

Fig. 10. Query Memory Footprint

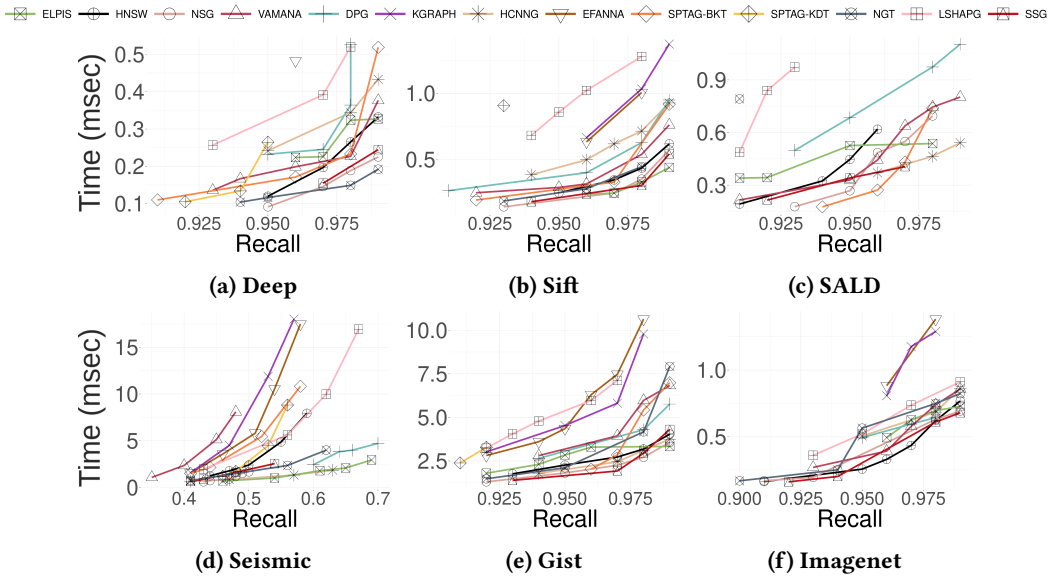
Fig. 11. Query Beam Width

HCNNG, and EFANNA) for the sake of clarity, as their search was significantly slower than the best baselines. Full results are in [3]. Other methods were omitted from the 100GB and 1B dataset sizes due to various limitations. The indexes for SPTAG and NGT could not be built on the larger datasets within 48 hours. EFANNA was excluded due to its high footprint, and likewise for methods based on it such as NSG and SSG. Finally, KGraph, DPG, and LSHAPG were excluded due to unsatisfactory results on 1M and 25GB.

Query Memory Footprint and Beam Width. Figure 10 indicates that Vamana, followed by ELPIS, have the lowest memory footprint during search. Even though ELPIS has a smaller index size, it adopts a contiguous memory storing during search, which increases the index footprint when loaded into memory. Besides, Figure 11 shows that Elpis requires the smallest beam width to reach similar query accuracy. Having a very high beam width indicates that the beam search requires to visit a wider area to and making more distance calculations to retrieves the NN answers.

Real Datasets. On datasets with 1M vectors (Figure 12), ELPIS and NSG/SSG perform best on Sift1M, achieving the highest performance for 0.99 and lower recall, respectively. For Seismic1M, HCNNG and ELPIS share the top rank. NGT, SSG, and NSG excel on Deep1M, while HCNNG leads on SALD1M at the highest recall, followed by SPTAG and NSG at lower recall levels. In ImageNet1M, NSG/SSG and HNSW rank as the top performers. Across most scenarios, SSG and NSG show similar performance. However, LSHAPG demonstrates limitations, requiring more computation to achieve high accuracy. Its probabilistic rooting prunes promising neighbors, requiring a larger beam width and tighter L-bsf lower-bound distance during search.When moving to 25GB datasets (Figure 13), SSG, NSG, NGT and HCNNG experience a drop in performance, and ELPIS takes the lead with the best overall performance together with SPTAG-BKT for SALD25GB .It is worth noting that none of the methods achieved an accuracy over 0.8 on the Seismic dataset, leading us to report results for these lower recall values. The significant indexing footprint of NSG prevented us from extending its evaluation to larger datasets, as constructing the EFANNA graph (which NSG depends on) requires more memory than the available 1.4TB. For hard query workloads in Figure 15 we compare the best-performing methods from the two most performing graph paradigms, ND-based and DC-based methods, including HNSW, NSG, ELPIS and SPTAG-BKT. SPTAG-BKT achieves the overall best performance for 1% noise query set, as we increase the noise up to 10%, SPTAG-BKT's performance deteriorates, which we can relate to SPTAG BKT structures failing to identify good seed points. At the same time, the other competitors gain an advantage, with ELPIS taking the lead. When analyzing very large datasets of 1 billion vectors, Figure 16 shows the superiority of ELPIS which is up to an order of magnitude faster at achieving 0.95 accuracy, thanks to its design that supports multi-threading for single query answering This trend is consistent across subsets ranging from 100GB (Figure 14) to 250GB (detailed results are reported in [3]).

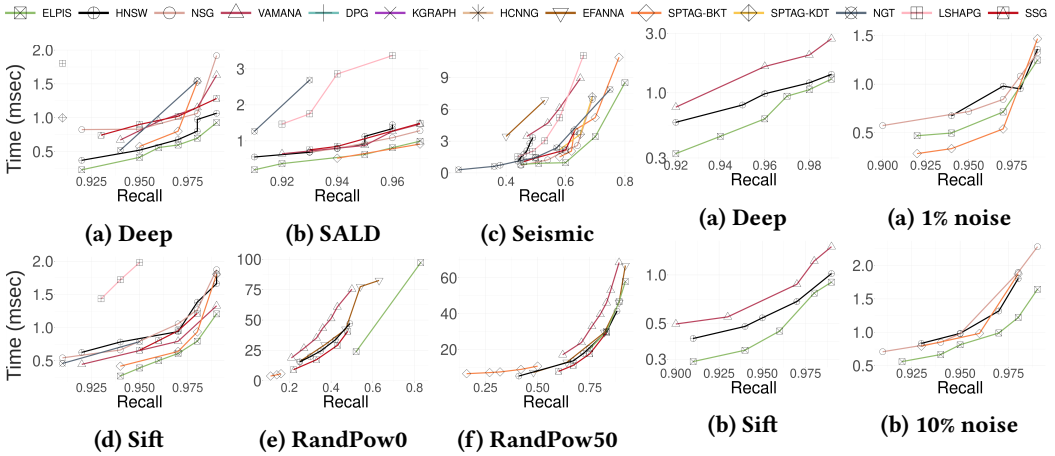
Data Distributions. We assess top performers representing different paradigms (EFANNA, Vamana, SSG, HNSW, ELPIS, and SPTAG-BKT) on challenging datasets (Fig. 4). Results (Figs. 13e and 13f) indicate that ELPIS consistently achieves high accuracy across skewness levels (0 to 50), outperforming other methods. As skewness increases, search becomes easier so most graph-based approaches improve but ELPIS maintains its superiority.



(a) Deep (b) Sift (c) SALD

(d) Seismic (e) Gist (f) Imagenet

Fig. 12. Query performance on 1M vectors



(a) Deep (b) SALD (c) Seismic

(d) Sift (e) RandPow0 (f) RandPow50

Fig. 13. 25GB datasets

(a) Deep

(b) Sift

Fig. 14. 100GB datasets

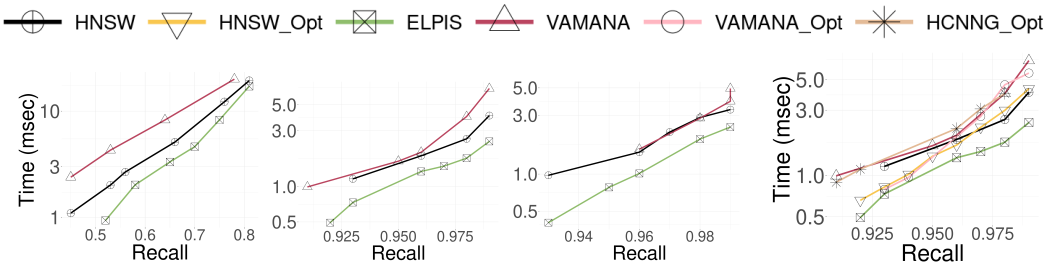
(a) 1% noise

(b) 10% noise

Fig. 15. Varying workloads

Implementation Impact. We evaluate the performance of original implementations of the best performing methods on 1B experiments, i.e., Vamana, HNSW, and ELPIS against optimized methods from the ParlayANN library [98] (Vamana_Opt, HNSW_Opt, and HCNNG_Opt) on Deep1B.

Figure 17 indicates that Vamana_Opt and HNSW_Opt are faster for recall below 0.97 compared to their original counterparts, due to more efficient data structures [98, 134]. However, at higher recall, this advantage diminishes as distance computations dominate; HCNNG_Opt is competitive with Vamana and HNSW, while ELPIS maintains a performance lead.



(a) Text2Image (b) Deep (c) Sift

Fig. 16. 1B Datasets

Fig. 17. Optimized Implementations (Deep1B)

5 Discussion

In the previous section, we presented the results of an extensive evaluation of twelve state-of-the-art graph-based vector search methods. Table 3 summarizes the evaluation across key criteria: for search, we assess efficiency, accuracy, and the number of tunable parameters; for indexing, we evaluate efficiency at high recall, memory footprint, and parameter tuning complexity.

The best-performing methods, HNSW, VAMANA, and ELPIS, have the best search performance and index efficiency. However, ELPIS requires an extra parameter during both indexing (leaf size) and search (nprobes), whereas VAMANA requires an extra parameter to tune during indexing (alpha). NSG and SSG exhibit efficient query performance, but their indexing capability is hindered because of their base graph EFANNA, which similarly to KGraph, is tedious to tune and suffers from high indexing time and footprint. Both SPTAG (BKT) and NGT show satisfactory performance during search; however, they do not scale well to large datasets and require more tuning compared to the best methods. The assessment of HCNNG is based on the optimized parlayNN implementation which has shown competitive performance on large-scale datasets.

We now summarize the key insights and pinpoint promising research directions.

Unexpected Results. Our results lead to interesting observations that warrant further study.
(1) Stacked NSW: while hierarchical layers of NSW graphs have shown promise in improving search performance on billion-scale datasets (Figure 6), our experiments demonstrate that a simpler approach like K-random sampling can achieve better results on smaller and medium-sized datasets.
(2) Scalability of Graph Approaches: while all graph-based vector search methods can efficiently build indexes on small datasets, most approaches face significant scalability challenges. Some methods (SPTAG, NGT, NSG, and SSG) demonstrate impressive search performance on 1M and 25GB datasets (Figs. 12b, 12a, 13c, 13a, and 13b) but their index construction could not scale to 100GB and billion-scale datasets. An important research direction is to improve the indexing scalability for these methods either by adopting summarization techniques during index construction or by using a scalable data structure to construct the base graph (i.e IVFPQ [73] to find the neighbors of nodes during insertion).
(3) DC-based approach for hard datasets and workloads: an interesting finding was the superior performance of DC-based approaches compared to other methods like HNSW, NSG, and Vamana

✓ Good ∼ Medium × Bad

Method	Query Answering			Index Building		
	Efficiency	Accuracy	Tuning	Efficiency	Footprint	Tuning
HNSW	✓	✓	✓	✓	✓	✓
ELPIS	✓	✓	∼	✓	✓	∼
VAMANA	✓	✓	✓	✓	✓	∼
NSG	✓	✓	✓	∼	∼	∼
SSG	✓	✓	✓	∼	∼	∼
EFANNA	×	∼	×	×	×	×
KGRAPH	×	×	×	×	×	×
DPG	×	∼	∼	∼	∼	∼
SPTAG	∼	✓	×	×	✓	×
HCNNG	✓	✓	✓	✓	✓	∼
LSHAPG	×	∼	×	∼	✓	✓
NGT	∼	∼	×	×	✓	×
SPTAG	∼	∼	∼	×	✓	×

Table 3. Comparative Analysis

on challenging datasets/workloads such as Seismic, RandPow0, RandPow50 and Deep hard query workload for 1M and 25GB dataset sizes. We believe the DC strategy helps in this context because the graphs are built on clustered subsets of data, which facilitates beam search in retrieving more accurate nearest neighbors (NN), as opposed to running the search on the entire dataset, which results in lower accuracy (Figures 12d, 13c, 13e, and 13f). The optimized implementations from [98] exhibit faster query times during search, but the gap narrows down with high recall.

Neighborhood Diversification. Adopting an ND strategy to sparsify the graph *always* leads to better search performance, particularly as the dataset size grows (Figures 12d, 13c). Our experiments also show that RND and MOND lead to the best search performance overall (Figure 5). Nevertheless, we believe there is significant room for improvement in this direction. Besides, we argue that further theoretical studies are necessary to better understand the trade-offs between proximity and sparsity, and thus build graph structures that are efficient during search and maintain good connectivity.

Seed Selection: Our experiments demonstrate that the SS strategy plays a crucial role in enhancing not only search performance (Figure 6) but also indexing efficiency (Table 2). An important research direction is to develop novel, lightweight SS strategies. Such strategies could significantly improve the overall performance of graph-based vector search, both in terms of indexing and query-answering. Additionally, they could enhance the ability to handle out-of-distribution queries, particularly for large datasets where efficient seed selection becomes even more critical (Figures 6c, 6f).

Data-Adaptive Techniques. Our experiments evaluate the performance of various graph-building paradigms within our taxonomy (SS, NP, II, ND, and DC). While NP-based methods perform the worst overall and are the least scalable, there is no clear winner across all dataset sizes and query workloads. *(1) Scalability:* II-based approaches have superior efficiency during indexing and higher scalability in both querying and indexing. *(2) Query Answering:* ND-based methods have the best query performance overall. Meanwhile, DC-based approaches are superior on challenging datasets (High LID&Low RC, Fig. 4) and hard query workloads (Fig. 16a, 12d, 13c, 15, 13f, 13e). A promising research direction would be to develop techniques that adapt to dataset characteristics such as dataset size, dimensionality, RC and LID to excel both in indexing and query answering across a variety of query workloads and dataset sizes.

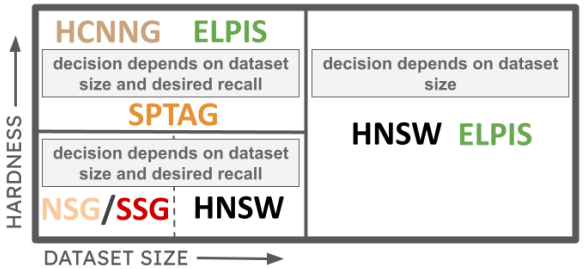


Fig. 18. Recommendations (Indexing + 10K queries)

Hybrid Design. Most recent methods use a mix of paradigms. HNSW leverages II to scale index construction to large datasets and ND to support efficient query answering. ELPIS incorporates a DC-based strategy during both index building and search to further enhance the scalability of HNSW across varying dataset difficulty levels. Interestingly, Vamana, relying only on the ND paradigm, achieves good search performance and scalability, however its indexing time is prohibitive. A promising research direction is building hybrid approaches that combine the key strengths of different techniques, particularly II, ND and DC. Besides, devising novel base graphs, clustering and summarization techniques tailored for DC-based methods can further improve their performance.
Optimized Libraries. Our experiments with the parlayNN library [98] (Fig. 17) indicate the importance of such work. For instance, the parlayNN HCNNG_Opt implementation was scalable to 1B datasets, whereas the non-optimized version could not scale beyond 25GB. Other methods could benefit from such optimizations to scale to large datasets, and further efforts are needed by the community to adopt and support libraries such as parlayNN.
Recommendations. Our study demonstrates varying performance trends across datasets of different sizes, query workloads of different hardness and desired recall values. Figure 18 provides recommendations for methods based on these criteria. For small to medium-sized datasets (25GB and below), HNSW, NSG, and its improved version, SSG, consistently demonstrate excellent performance on easier datasets (Fig.12a, 12b, 12f, 12e). On harder datasets, DC-based methods like SPTAG, ELPIS, and HCNNG prove more efficient (Figs. 12d 13c, 12c, 13b, 15a, 15b). On large datasets (100GB and above), HNSW and ELPIS consistently rank as top choices (Figs.14, 16).

6 Conclusions

In this paper, we conduct a survey of the SotA graph-based methods for in-memory ng-approximate vector search, proposing a new taxonomy based on five key design paradigms. The chronological development and inter-method influence are outlined, along with an evaluation of key design choices like seed selection and neighborhood diversification.

Through extensive experimentation on datasets with up to 1B vectors, we highlight the scalability challenges faced by most methods, with incremental insertion methods showing the best scalability on datasets exceeding 100GB. We observe that light-weight hierarchical structures help select better seeds to start the search on billion-scale datasets, and that neighborhood diversification is a key contributor in improving the query answering performance, with RND and MOND being the best techniques. We also propose promising research directions.

Acknowledgments

This work used the UM6P African Supercomputing Center (ASCC). Work partially supported by EU Horizon projects AI4Europe (101070000), TwinODIS (101160009), ARMADA (101168951), DataGEMS (101188416) and RECITALS (101168490).

References

[1] 2019. Hnswlib - fast approximate nearest neighbor search. https://github.com/nmslib/hnswlib.

[2] 2022. ELPIS Archive. https://github.com/scalablesimilaritysearch/ELPIS.

[3] 2024. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art (Archive). https://github.com/zeraph6/GASS_Repo.

[4] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. Code2vec: Learning Distributed Representations of Code. 3, POPL (2019).

[5] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2015. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 29–38.

[6] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. HD-index: Pushing the Scalability-accuracy Boundary for Approximate kNN Search in High-dimensional Spaces. *PVLDB* 11, 8 (2018), 906–919.

[7] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2017. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. In *SISAP*.

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

[9] Martin Aumüller and Matteo Ceccarello. 2021. The role of local dimensionality measures in benchmarking nearest neighbor search. *Inf. Syst.* 101 (2021), 101807. doi:10.1016/J.IS.2021.101807

[10] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. 2013. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company.

[11] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. Elpis: Graph-Based Similarity Search for Scalable Data Science. *PVLDB* 16, 6 (2023).

[12] A. Babenko and V. Lempitsky. 2015. The Inverted Multi-Index. *TPAMI* 37, 6 (2015).

[13] Dmitry Baranchuk and Artem Babenko. 2021. Text-to-Image dataset for billion-scale similarity search. https://research.yandex.com/datasets/text-to-image-dataset-for-billion-scale-similarity-search.

[14] Olivier Beaumont, Anne-Marie Kermarrec, Loris Marchal, and Étienne Rivière. 2007. VoroNet: A scalable object network based on Voronoi tessellations. In *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 1–10.

[15] Olivier Beaumont, Anne-Marie Kermarrec, and Étienne Rivière. 2007. Peer to peer multidimensional overlays: Approximating complex structures. In *International Conference On Principles Of Distributed Systems*. Springer, 315–328.

[16] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. In *INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*. ACM, 322–331.

[17] Jeffrey S Beis and David G Lowe. 1997. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. IEEE, 1000–1006.

[18] Andreas Blattmann, Robin Rombach, Kaan Oktay, Jonas Müller, and Björn Ommer. 2022. Retrieval-augmented diffusion models. *Advances in Neural Information Processing Systems* 35 (2022).

[19] Paul Boniol, Michele Linardi, Federico Roncallo, and Themis Palpanas. 2020. Automated Anomaly Detection in Large Sequences. In *ICDE*.

[20] Paul Boniol and Themis Palpanas. 2020. Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series. *PVLDB* (2020).

[21] Sébastien Bubeck and Ulrike von Luxburg. 2009. Nearest Neighbor Clustering: A Baseline Method for Consistent Clustering with Arbitrary Objective Functions. *JMLR* 10 (2009).

[22] Simon Byers and Adrian E. Raftery. 1998. Nearest-Neighbor Clutter Removal for Estimating Features in Spatial Point Processes. *JASA* 93, 442 (1998).

[23] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. 2014. Beyond One Billion Time Series: Indexing and Mining Very Large Time Series Collections With *i*SAX2+. *Knowledge and information systems* 39, 1 (2014), 123–151.

[24] Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. 2002. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *ACM Trans. Database Syst.* 27, 2 (June 2002), 188–228. doi:10.1145/568518.568520

[25] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng. 2023. Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. *Proc. VLDB Endow.* (2023).

[26] Georgios Chatzigeorgakidis, Dimitrios Skoutas, Kostas Patroumpas, Themis Palpanas, Spiros Athanasiou, and Spiros Skiadopoulos. 2019. Local Pair and Bundle Discovery over Co-Evolving Time Series. In *Proceedings of the 16th*

International Symposium on Spatial and Temporal Databases, SSTD. ACM, 160–169. doi:10.1145/3340964.3340982

[27] Georgios Chatzigeorgakidis, Dimitrios Skoutas, Kostas Patroumpas, Themis Palpanas, Spiros Athanasiou, and Spiros Skiadopoulos. 2019. Local Similarity Search on Geolocated Time Series Using Hybrid Indexing. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL*. ACM, 179–188. doi:10.1145/3347146.3359349

[28] Georgios Chatzigeorgakidis, Dimitrios Skoutas, Kostas Patroumpas, Themis Palpanas, Spiros Athanasiou, and Spiros Skiadopoulos. 2023. Efficient Range and kNN Twin Subsequence Search in Time Series. *IEEE Trans. Knowl. Data Eng.* 35, 6 (2023), 5794–5807. doi:10.1109/TKDE.2022.3167257

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

[30] Paolo Ciaccia and Marco Patella. 2000. PAC Nearest Neighbor Queries: Approximate and Controlled Search in High-Dimensional and Metric Spaces. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, David B. Lomet and Gerhard Weikum (Eds.). IEEE Computer Society, 244–255. doi:10.1109/ICDE.2000.839417

[31] Daniel Croft, Manish Gupta, Vlad Josifovski, Parikshit Narayanan, Weitian Wu, and Yan Xue. 2021. DiskANN: Fast Approximate Nearest Neighbor Search on Disk. GitHub repository. https://github.com/microsoft/DiskANN Accessed: 2024-10-25.

[32] Sanjoy Dasgupta and Yoav Freund. 2008. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 537–546.

[33] DBAIWangGroup. 2023. NNS Benchmark: Evaluating Approximate Nearest Neighbor Search Algorithms in High Dimensional Euclidean Space - DPG Algorithm. https://github.com/DBAIWangGroup/nns_benchmark/tree/master/algorithms/DPG. GitHub repository.

[34] David P Dobkin, Steven J Friedman, and Kenneth J Supowit. 1990. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry* 5, 4 (1990), 399–407.

[35] W. Dong. 2022. Kgraph, an open source library for k-nn graph construction and nearest neighbor search. www.kgraph.org.

[36] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*. 577–586.

[37] Sumeet Dua and Xian Du. 2011. *Data Mining and Machine Learning in Cybersecurity* (1st ed.). Auerbach Publications, USA.

[38] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *VLDBJ* 11, 11 (2018).

[39] Karima Echihabi. 2019. Truly Scalable Data Series Similarity Search. In *Proceedings of the VLDB 2019 PhD Workshop (CEUR Workshop Proceedings, Vol. 2399)*. CEUR-WS.org.

[40] Karima Echihabi. 2020. High-Dimensional Similarity Search: From Time Series to Deep Network Embeddings. In *SIGMOD*.

[41] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2022. Hercules Against Data Series Similarity Search. *PVLDB* 15, 10 (2022).

[42] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2020. Scalable Machine Learning on High-Dimensional Vectors: From Data Series to Deep Network Embeddings. In *WIMS 2020: The 10th International Conference on Web Intelligence, Mining and Semantics*. ACM, 1–6.

[43] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. High-Dimensional Similarity Search for Scalable Data Science *(ICDE)*.

[44] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2018. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *PVLDB* 12, 2 (2018).

[45] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2019. Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. *PVLDB* (2019).

[46] Herbert Edelsbrunner. 2012. *Algorithms in Combinatorial Geometry* (1st ed.). Springer Publishing Company, Incorporated.

[47] Andreas Engel, Rémi Monasson, and Alexander K Hartmann. 2004. On large deviation properties of erdös–rényi random graphs. *Journal of Statistical Physics* 117, 3 (2004), 387–426.

[48] Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and George Paterakis. 2023. FreSh: A Lock-Free Data Series Index. In *SRDS*.

[49] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2000. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the ninth international conference on Information and knowledge management*. 202–209.

[50] Incorporated Research Institutions for Seismology with Artificial Intelligence. 2018. Seismic Data Access. http://ds.iris.edu/data/access/.
[51] Steven Fortune. 1995. Voronoi diagrams and Delaunay triangulations. *Computing in Euclidean geometry* (1995), 225–265.
[52] Cong Fu and Deng Cai. 2016. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228* (2016).
[53] Cong Fu, Changxu Wang, and Deng Cai. 2021. High Dimensional Similarity Search with Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).
[54] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474.
[55] Miao Fu, Huizhong Cai, Zhiwei Fu, Jie Tang, and Zhiyuan Liu. 2019. Efficient and Effective Approximate Nearest Neighbor Search. GitHub repository. https://github.com/Yotta-Chen/NSG Accessed: 2024-10-25.
[56] K Ruben Gabriel and Robert R Sokal. 1969. A new statistical approach to geographic variation analysis. *Systematic zoology* 18, 3 (1969), 259–278.
[57] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.
[58] Anna Gogolou, Theophanis Tsandilas, Karima Echihabi, Themis Palpanas, and Anastasia Bezerianos. 2020. Data Series Progressive Similarity Search with Probabilistic Quality Guarantees. In *SIGMOD*.
[59] Anna Gogolou, Theophanis Tsandilas, Themis Palpanas, and Anastasia Bezerianos. 2019. Progressive Similarity Search on Time Series Data. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference*.
[60] R. M. Gray and D. L. Neuhoff. 2006. Quantization. *IEEE Trans. Inf. Theor.* 44, 6 (Sept. 2006), 2325–2383. doi:10.1109/18.720541
[61] Michael Günther, Maik Thiele, and Wolfgang Lehner. 2019. RETRO: Relation Retrofitting For In-Database Machine Learning on Textual Data. *arXiv preprint arXiv:1911.12674* (2019).
[62] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*.
[63] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.
[64] Junfeng He, Sanjiv Kumar, and Shih-Fu Chang. 2012. On the difficulty of nearest neighbor search. *arXiv preprint arXiv:1206.6411* (2012).
[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[66] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware Locality-sensitive Hashing for Approximate Nearest Neighbor Search. *PVLDB* 9, 1 (2015), 1–12.
[67] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality *(STOC)*.
[68] Masajiro Iwasaki. 2016. Pruned bi-directed k-nearest neighbor graph for proximity search. In *International Conference on Similarity Search and Applications*. Springer, 20–33.
[69] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A survey on locality sensitive hashing algorithms and their applications. *arXiv preprint arXiv:2102.08942* (2021).
[70] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 33. IEEE, 117–128.
[71] H. Jegou, R. Tavenard, M. Douze, and L. Amsaleg. 2011. Searching in one billion vectors: Re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 861–864. doi:10.1109/ICASSP.2011.5946540
[72] Zhongming Jin, Debing Zhang, Yao Hu, Shiding Lin, Deng Cai, and Xiaofei He. 2014. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics* 44, 11 (2014), 2167–2177.
[73] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
[74] William Johnson and Joram Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*. Contemporary Mathematics, Vol. 26. American Mathematical Society, 189–206.
[75] Kari Karhunen. 1947. *Ueber lineare Methoden in der Wahrscheinlichkeitsrechnung*. Soumalainen Tiedeakatemia.
[76] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906* (2020).
[77] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems* 3, 3 (2001), 263–286.

doi:10.1007/PL00011669

[78] Eamonn Keogh, Jessica Lin, and Ada Fu. 2005. HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence. In *ICDM*.

[79] Jon Kleinberg et al. 2002. Small-world phenomena and the dynamics of information. *Advances in neural information processing systems* 1 (2002), 431–438.

[80] Jon M Kleinberg. 2000. Navigation in a small world. *Nature* 406, 6798 (2000), 845–845.

[81] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: A Scalable Bottom-Up Approach for Building Data Series Indexes. *Proc. VLDB Endow.* 11, 6 (2018), 677–690. doi:10.14778/3184470.3184472

[82] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2019. Coconut: Sortable Summarizations for Scalable Indexes over Static and Streaming Data Series. *VLDBJ* 28, 6 (2019).

[83] Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. 2020. Pre-training via paraphrasing. *Advances in Neural Information Processing Systems* 33 (2020), 18470–18481.

[84] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[85] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. 2019. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *TKDE* (2019).

[86] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data: experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.

[87] Jessica Lin, Eamonn J. Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD, San Diego, California, USA*.

[88] Peng-Cheng Lin and Wan-Lei Zhao. 2019. Graph based nearest neighbor search: Promises and failures. *arXiv preprint arXiv:1904.02077* (2019).

[89] Michele Linardi and Themis Palpanas. 2018. Scalable, Variable-Length Similarity Search in Data Series: The ULISSE Approach. *Proc. VLDB Endow.* 11, 13 (2018), 2236–2248. doi:10.14778/3275366.3284968

[90] Michele Linardi and Themis Palpanas. 2020. Scalable Data Series Subsequence Matching with ULISSE. *VLDBJ* (2020).

[91] G. Linden, B. Smith, and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (2003).

[92] Michel Loeve. 1948. Functions aleatoires du second ordre. *Processus stochastique et mouvement Brownien* (1948), 366–420.

[93] Kejing Lu. 2023. HVS: Hierarchical Graph Structure Based on Voronoi Diagrams for Solving Approximate Nearest Neighbor Search. https://github.com/Kejing-Lu/hvs

[94] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 15, 2 (2021), 246–258.

[95] Mikko I Malinen and Pasi Fränti. 2014. Balanced k-means for clustering. In *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop, S+ SSPR 2014, Joensuu, Finland, August 20-22, 2014. Proceedings*. Springer, 32–41.

[96] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[97] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.

[98] Magdalen Dobson Manohar, Zheqi Shen, Guy Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. 2024. ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search Algorithms. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 270–285.

[99] David W Matula and Robert R Sokal. 1980. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical analysis* 12, 3 (1980), 205–222.

[100] Stanislav Morozov and Artem Babenko. 2018. Non-metric similarity graphs for maximum inner product search. *Advances in Neural Information Processing Systems* 31 (2018).

[101] Marius Muja and David G. Lowe. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP International Conference on Computer Vision Theory and Applications*. 331–340.

[102] Javier Vargas Munoz, Marcos A Gonçalves, Zanoni Dias, and Ricardo da S Torres. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition* 96 (2019), 106970.

[103] Bilegsaikhan Naidan, Leonid Boytsov, and Eric Nyberg. 2015. Permutation Search Methods Are Efficient, Yet Faster Search is Possible. *PVLDB* 8, 12 (2015), 12 pages. doi:10.14778/2824032.2824059
[104] Mark EJ Newman. 2005. Power laws, Pareto distributions and Zipf's law. *Contemporary physics* 46, 5 (2005), 323–351.
[105] Trong Duc Nguyen, Anh Tuan Nguyen, and Tien N. Nguyen. 2016. Mapping API Elements for Code Migration with Vector Representations. In *ICSE*.
[106] Aude Oliva and Antonio Torralba. 2001. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision* 42 (2001), 145–175.
[107] Themis Palpanas. 2015. Data series management: The road to big sequence analytics. *ACM SIGMOD Record* 44, 2 (2015), 47–52.
[108] Themis Palpanas. 2020. Evolution of a Data Series Index: the iSAX Family of Data Series Indexes. *Communications in Computer and Information Science (CCIS)* 1197 (2020).
[109] Themis Palpanas and Volker Beckmann. 2019. Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA). *ACM SIGMOD Record* 48, 3 (2019).
[110] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2020. Messi: In-memory data series indexing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 337–348.
[111] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. Fast Data Series Indexing for In-Memory Data. *VLDBJ* (2021).
[112] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. SING: Sequence Indexing Using GPUs. In *ICDE*.
[113] Botao Peng, Themis Palpanas, and Panagiota Fatourou. 2020. ParIS+: Data Series Indexing on Multi-core Architectures. *TKDE* (2020).
[114] François Petitjean, Germain Forestier, Geoffrey I. Webb, Ann E. Nicholson, Yanping Chen, and Eamonn J. Keogh. 2014. Dynamic Time Warping Averaging of Time Series Allows Faster and More Accurate Classification. In *ICDM*.
[115] Alexander Ponomarenko, Yury Malkov, Andrey Logvinov, and Vladimir Krylov. 2011. Approximate nearest neighbor search small world approach. In *International Conference on Information and Communication Technologies & Applications*, Vol. 17.
[116] William Pugh. 1990. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM* 33, 6 (1990), 668–676.
[117] Python API. 2022. openmc.stats.PowerLaw. https://docs.openmc.org/en/stable/pythonapi/generated/openmc.stats.PowerLaw.html.
[118] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient Algorithms for Mining Outliers from Large Data Sets. *SIGMOD Rec.* 29, 2 (2000).
[119] D Raj Reddy et al. 1977. Speech understanding systems: A summary of results of the five-year research effort. *Department of Computer Science. Camegie-Mell University, Pittsburgh, PA* 17 (1977), 138.
[120] Microsoft Research. 2018. SPTAG: A Library for Fast Approximate Nearest Neighbor Search. GitHub repository. https://github.com/microsoft/SPTAG Accessed: 2024-10-25.
[121] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115 (2015), 211–252.
[122] Patrick Schäfer and Mikael Högqvist. 2012. SFA: A Symbolic Fourier Approximation and Index for Similarity Search in High Dimensional Datasets. In *Proceedings of the 15th International Conference on Extending Database Technology (EDBT '12)*.
[123] Roger W Schvaneveldt, Francis T Durso, and Donald W Dearholt. 1989. Network structures in proximity data. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 249–284.
[124] Michael Ian Shamos and Dan Hoey. 1975. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. IEEE, 151–162.
[125] Lei Shi. 2013. Trading-off among accuracy, similarity, diversity, and long-tail: a graph-based recommendation approach. In *Proceedings of the 7th ACM Conference on Recommender Systems*. 57–64.
[126] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2022. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. *CoRR* abs/2205.03763 (2022). doi:10.48550/arXiv.2205.03763 arXiv:2205.03763
[127] Skoltech Computer Vision. 2018. Deep billion-scale indexing. http://sites.skoltech.ru/compvision/noimi.
[128] Liuyihan Song, Pan Pan, Kang Zhao, Hao Yang, Yiming Chen, Yingya Zhang, Yinghui Xu, and Rong Jin. 2020. Large-scale training system for 100-million classification at alibaba. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2909–2930.
[129] Suhas Jayaram Subramanya, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 13766–13776.

[130] Kohei Sugawara, Hayato Kobayashi, and Masajiro Iwasaki. 2016. On approximately searching for similar word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2265–2275.
[131] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: Solving c-approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. *PVLDB* 8, 1 (2014).
[132] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proceedings of the VLDB Endowment* (2014).
[133] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2009. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 563–576.
[134] The Parlayann Team. 2023. Parlayann: A deep learning library for parallel computation. https://github.com/parlayann/parlayann. Accessed: 2024-10-25.
[135] TEXMEX Research Team. 2018. Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr/.
[136] Godfried T Toussaint. 1980. The relative neighbourhood graph of a finite planar set. *Pattern recognition* 12, 4 (1980), 261–268.
[137] Godfried T Toussaint. 2002. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface* 34 (2002).
[138] Southwest University. 2018. Southwest University Adult Lifespan Dataset (SALD). http://fcon_1000.projects.nitrc.org/indi/retro/sald.html?utm_source=newsletter&utm_medium=email&utm_content=See20Data&utm_campaign=indi-1.
[139] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-Scale Commodity Embedding for E-Commerce Recommendation in Alibaba. In *KDD*.
[140] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-nn graph construction for visual descriptors. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1106–1113.
[141] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. 2013. Trinary-projection trees for approximate nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 36, 2 (2013), 388–403.
[142] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 14, 11 (jul 2021), 1964–1978. doi:10.14778/3476249.3476255
[143] Qitong Wang, Ioana Ileana, and Themis Palpanas. 2025. LeaFi: Data Series Indexes on Steroids with Learned Filters. *Proc. ACM Manag. Data* (2025).
[144] Qitong Wang and Themis Palpanas. 2021. Deep Learning Embeddings for Data Series Similarity Search. In *SIGKDD*.
[145] Qitong Wang and Themis Palpanas. 2023. SEAnet: A Deep Learning Architecture for Data Series Similarity Search. *TKDE* (2023).
[146] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. 2013. A Data-adaptive and Dynamic Segmentation Index for Whole Matching on Time Series. *PVLDB* 6, 10 (2013).
[147] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph- and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions. *IEEE Data Eng. Bull.* 46, 3 (2023), 3–21.
[148] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A Compact and Adaptive Index for Large Data Series Collections. In *ACM SIGMOD*.
[149] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2024. DumpyOS: A data-adaptive multi-ary index for scalable data series similarity search. *The VLDB Journal* (2024), 1–25.
[150] T. Warren Liao. 2005. Clustering of time series data—a survey. *Pattern Recognition* 38, 11 (2005).
[151] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.
[152] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. VLDB*. 194–205.
[153] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 9 (2024), 2241–2254.
[154] K. Williams, L. Li, M. Khabsa, J. Wu, P. C. Shih, and C. L. Giles. 2014. A Web Service for Scholarly Big Data Information Extraction. In *ICWS*.
[155] Yan Xia, Kaiming He, Fang Wen, and Jian Sun. 2013. Joint Inverted Indexing. *ICCV* (2013).
[156] D. E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. 2017. DPiSAX: Massively Distributed Partitioned iSAX. In *ICDM*.
[157] Djamel-Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2020. Massively Distributed Time Series Indexing and Querying. *TKDE 31(1)* (2020).

[158] Yahoo Japan Corporation. 2022. NGT: Neighborhood Graph and Tree for High-dimensional Data. https://github.com/yahoojapan/NGT. Accessed: 2024-10-20.
[159] Dmitry Yarotsky. 2019. Neighborhood Graph and Tree for Indexing High-dimensional Data. GitHub repository. https://github.com/yahoojapan/NGT Accessed: 2024-10-25.
[160] Peter N Yianilos. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Soda*, Vol. 93. 311–21.
[161] Yan-Ming Zhang, Kaizhu Huang, Guanggang Geng, and Cheng-Lin Liu. 2013. Fast kNN graph construction with locality sensitive hashing. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 660–674.
[162] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 635–644.
[163] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1979–1991.
[164] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. 2016. LSH ensemble: internet-scale domain search. *PVLDB* 9, 12 (2016), 1185–1196.
[165] Yifan Zhu, Miao Fu, Huizhong Cai, Zhiwei Fu, Jie Tang, and Zhiyuan Liu. 2020. Scalable Sparse Graphs for Efficient Approximate Nearest Neighbor Search. GitHub repository. https://github.com/ZJULearning/SSG Accessed: 2024-10-25.
[166] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2016. ADS: the adaptive data series index. *VLDB J.* (2016).
[167] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. 2018. Generating Data Series Query Workloads. *The VLDB Journal* 27, 6 (Dec. 2018), 823–846. doi:10.1007/s00778-018-0513-x

Received July 2024; revised September 2024; accepted November 2024