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

Ilias Azizi [1] **Karima Echihabi** [2] **Themis Palpanas** [3] **Vassilis Christophides** [1]

Abstract

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. Although several paradigms (seed selection, incremental insertion, neighborhood propagation, neighborhood diversification, and divide-and-conquer) have been employed to design in-memory graph-based vector search algorithms, a systematic comparison of the key algorithmic advances is still missing. 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. An extended version of this work appeared in ACM SIGMOD 2025.

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. The volume and dimensionality of this data, which can exceed multiple terabytes and thousands of dimensions, make its analysis very challenging. A critical component of these data analysis tasks is vector search (Echihabi et al., 2021; Palpanas, 2015). It supports recommendation (Wang et al., 2018), information retrieval (Williams et al., 2014), clustering (Bubeck & von Luxburg, 2009), classification (Petitjean et al., 2014), entity resolution (Christophides et al., 2020), outlier detection (Boniol & Palpanas, 2020) and AI explainability (Kaneko, 2023) in many fields including bioinformatics, computer vision, finance. More recently, vector search has been playing a crucial role in improving the performance and interpretability of Large Language Models and reducing their hallucinations (Blattmann et al., 2022; Karpukhin et al., 2020).

As these applications scale to ever-larger datasets, often terabytes in size and thousands of dimensions, vector search becomes not only a computational bottleneck, but also a significant contributor to system latency and energy consumption (Huang et al., 2024). In many AI pipelines, especially those deployed at inference time, similarity search dominates runtime cost (Desislavov et al., 2023). Optimizing vector search is therefore essential to building applications that are not only faster and more responsive, but also more efficient in terms of operational resources and environmental impact (Chiang et al., 2025).

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, with time complexity $O(nd)$—which becomes impractical and energy-intensive for large, high-dimensional datasets. To address this, state-of-the-art methods reduce the dimensionality d via compact representations and/or minimize the number of comparisons n through efficient indexing and pruning strategies. These improvements not only accelerate search but also reduce memory usage and computational overhead, contributing to more energy-efficient systems. Some approaches compute exact answers, while others ϵ- and δ-ϵ-approximate answers, with deterministic and probabilistic guarantees on the accuracy of the answers, or ng-approximate answers without any theoretical guarantees, but high accuracy in practice (Echihabi et al., 2019).

[1]ETIS UMR-8051 Laboratory, CY Cergy Paris Université, ENSEA, CNRS, France [2]College of Computer Science, University Mohammed VI Polytechnic, Morocco [3]LIPADE, Université Paris Cité - IUF, France. Correspondence to: Ilias Azizi <ilias.azizi@ensea.fr>.

Proceedings of the 1st Workshop on Vector Databases at International Conference on Machine Learning, 2025.

A large body of work has been dedicated to approximate vector search, which trades off accuracy for efficiency. These approaches are based on scans (Weber et al., 1998; Ferhatosmanoglu et al., 2000), trees (Camerra et al., 2014; Wang et al., 2025), graphs (Malkov & Yashunin, 2020; Subramanya et al., 2019; Munoz et al., 2019), inverted indexes (Jégou et al., 2011; Babenko & Lempitsky, 2015), hashing (Sun et al., 2014a;b), or a combination of these data structures (Chen et al., 2018; Azizi et al., 2023). 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 (Song et al., 2020; Johnson et al., 2019).

Graph-based methods typically represent a dataset S as a proximity graph $G(V, E)$, where V is the set of vertices corresponding to the n data points, and E is the set of edges connecting similar points. Query answering for a given query node V_Q usually involves running an ng-approximate beam search, starting with an initial set of seed nodes to warm up the candidate priority queue, followed by a greedy graph traversal expanding promising candidates. State-of-the-art (SotA) graph-based methods share this beam-search approach but differ mainly in their graph construction strategies and seed selection mechanisms.

Although several paradigms (seed selection, incremental insertion, neighborhood propagation, neighborhood diversification, and divide-and-conquer) have been employed to design in-memory graph-based vector search algorithms, a systematic comparison of the key algorithmic advances is still missing. The only empirical study dedicated entirely to this family of techniques is (Wang et al., 2021) without providing conclusive results due to the small scale of datasets used in experiments, not exceeding 1M vectors. As we will show in Section 3 that trends change as dataset size increases. Moreover, existing benchmarks (Aumüller et al., 2017; Simhadri et al., 2022) for evaluating vector search methods do not focus on graph-based approaches, and thus, do not shed light on why the best methods have superior query performance. To the best of our knowledge, our work is the first proposing a taxonomy of algorithmic advances grounded in the actual indexing and search design principles, rather than on abstract graph categories. This enables new insights into how key design choices impact the indexing and search performance across varying dataset sizes, addressing limitations in prior studies.

In this paper, we make the following contributions[1]:

• We identify five core paradigms underlying graph-based vector search: Seed Selection (SS), Neighborhood Propagation (NP), Incremental Insertion (II), Neighborhood Diversification (ND), and Divide-and-Conquer (DC). We briefly overview these paradigms and focus deeply on SS and ND, due to their significant impact on performance.

• We propose a taxonomy categorizing existing methods according to these paradigms, highlighting their chronological evolution and influence.

• We briefly survey state-of-the-art methods, discussing their design principles, strengths, and limitations.

• We conduct an extensive evaluation of twelve state-of-the-art methods on synthetic and real-world datasets (up to 1 billion vectors). Our experiments validate conventional wisdom, such as the superiority of incremental insertion methods in query performance and scalability (Azizi et al., 2023; Wang et al., 2021). However, we also find differences from recent studies (Wang et al., 2021), notably demonstrating better-than-previously-reported efficiency of SPTAG-BKT (Chen et al., 2018) on small datasets and superior indexing efficiency of HNSW (Malkov & Yashunin, 2020). Contrary to prior findings (Azizi et al., 2023), we also show that Vamana (Subramanya et al., 2019) is competitive on large scale datasets.

• We provide novel insights, such as (i) identifying the most effective ND technique for large datasets, and (ii) analyzing the impact of different SS methods on query and indexing performance on various dataset scales.

• Finally, we highlight promising research directions, including developing scalable NP and ND graph structures, data-adaptive seed selection techniques, theoretical analysis of ND methods, and tailored strategies (e.g., clustering, diversification) for DC-based approaches.

2. 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.

2.1. 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* (Shamos & Hoey, 1975). A proximity graph can be constructed using different distances such as the dot product (Morozov & Babenko, 2018), nevertheless the Euclidean distance remains the most popular one (Edelsbrunner, 2012). One of the earliest proximity graphs in the literature is the Delaunay Graph (*DG*). It is a planar dual

[1]Full paper appeared as (Azizi et al., 2025).

Algorithm 1 Beam Search (G, q, s, k, l)

Input: Graph G, query vector q, initial seeds s, result size k, beam width $l \geq k$
Output: k approximate nearest neighbors to q
Initialize candidate set $C \leftarrow s$
Initialize visited list $V \leftarrow \emptyset$
while $C \setminus V \neq \emptyset$ **do**
  $p^* \leftarrow \text{argmin}_{x_i \in C \setminus V}\, dist(q, x_i)$
  $C \leftarrow C \cup N_{out}(p^*)$
  $V \leftarrow V \cup \{p^*\}$
  if $|C| > l$ **then**
    Retain only the closest L points in C to q
  end if
end while
Return the closest k candidates in C to q

graph for the Voronoi Diagram (Fortune, 1995), 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 (Aurenhammer et al., 2013). A beam search (Reddy et al., 1977) (Algorithm 1) on a *DG* can find the exact nearest neighbors (Dobkin et al., 1990) 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 (Dobkin et al., 1990). Thus, SotA methods build alternative graph structures and use beam search to support efficient query answering (Gabriel & Sokal, 1969; Matula & Sokal, 1980; Toussaint, 2002).

2.2. 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 3).

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 (Dong et al., 2011). 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 (Beaumont et al., 2007a), with the idea of adding long-range links being inspired from Kleinberg's small-world model (Kleinberg, 2000; Kleinberg et al., 2002), with the difference that the latter selects the long-range links randomly.

Neighborhood Diversification (ND) was first introduced by the Relative Neighborhood Graph (RNG) (Toussaint, 1980). 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 (Chen et al., 2018) and HCNNG (Munoz et al., 2019) combine the individual graphs into one large graph, on which a beam search is performed, while ELPIS (Azizi et al., 2023) maintains the graphs separate and searches them in parallel.

2.3. Seed Selection

While state-of-the-art graph-based vector search methods vary in graph construction strategies, nearly all rely on beam search (Algorithm 1) for query answering, as it efficiently retrieves good answers in well-connected graphs. However, the choice of initial nodes—referred to as *seeds*—significantly impacts search efficiency: better seeds lead to fewer visited nodes and faster searches. Typically, an entry node or multiple candidate entry nodes are used to warm the beam search's priority queue; in the case of multiple seeds, the search picks the one closest to the query as the initial node, retaining others in the priority queue. While several methods propose additional indexes built on data samples to facilitate seed selection, none provide comprehensive evidence supporting their effectiveness. In this paper, we systematically examine the seed-selection techniques proposed in the literature:

(1) **Stacked-NSW (SN)**: HNSW (Malkov & Yashunin, 2020), inspired by skip lists (Pugh, 1990), builds hierar-

chical NSW graphs by sampling nodes from lower layers. Nodes are assigned a top layer $L = -\ln(\xi)/\ln(M/2)$, with $\xi \sim U(0,1)$ and out-degree M. Queries descend greedily from a fixed top-layer entry point, returning the closest sampled node to the query as entry point.

(2) **K-D Trees (KD)**: Constructs a single or multiple K-D Tree(s) (Beis & Lowe, 1997) on a dataset sample. During search,retrieves the set of seed points is retrieved through depth-first search traversal on the K-D Tree structure(s).

(3) **LSH**: Constructs an LSH index on a sample of the dataset to retrieve the seeds during search.

(4) **Medoid (MD)**: Uses medoid node as seed and entry point during query answering.

(5) **Single Fixed Random Entry Point (SF)**: A random node is selected and fixed as the entry point for all searches.

(6) **K-Sampled Random Seeds (KS)**: For each query, k random nodes are selected as seed points.

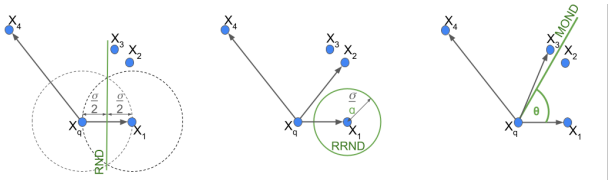
(7) **Balanced K-means Trees (KM)**: Constructs Balanced K-means Trees (BKT) (Malinen & Fränti, 2014) on a dataset sample. During search, seed points are retrieved via depth-first search on the BKT structure(s).

2.4. Neighborhood Diversification

Neighborhood Diversification (ND) builds sparse graphs by selectively pruning edges to balance short and long-range connections, thus reducing redundant comparisons. Initially proposed by RNG (Toussaint, 1980; 2002), ND was adapted for approximate vector search by several graph-based methods. Three main ND strategies exist: Relative Neighborhood Diversification (RND) used by HNSW (Malkov & Yashunin, 2020), NSG (Fu et al., 2019), SPTAG (Chen et al., 2018) and ELPIS (Azizi et al., 2023); Relaxed RND (RRND) introduced by Vamana (Subramanya et al., 2019); and Maximum-Oriented ND (MOND), proposed by DPG (Li et al., 2019) and NSSG (Fu et al., 2021). RND connects node X_q to candidate X_j only if no current neighbor X_i is closer to X_j. RRND relaxes this condition by factor $\alpha \geq 1$, reducing pruning and increasing edges. MOND selects edges based on an angle threshold $\theta \geq 60°$, favoring diverse edge orientations. RRND and MOND prune fewer edges than RND, thus resulting in denser graphs (cf. Figure 1). Note that any nodes pruned by RRND and MOND will eventually be pruned by RND, but not vice versa. Refer to (url, 2025) for a detailed proof.

2.5. A Taxonomy

Figure 2 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 2.4). 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).



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

Figure 1. Neighborhood diversification approaches

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 (Dong, 2022) 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 (Jin et al., 2014) and EFANNA (Fu & Cai, 2016). In parallel, NSW (Ponomarenko et al., 2011) introduced the II strategy for graph construction.

HNSW (Malkov & Yashunin, 2020) and DPG (Li et al., 2019) leveraged ND to enhance NSW and KGraph, respectively. The good performance of HNSW and DPG encouraged more methods to adopt the ND paradigm, including NGT (Yahoo Japan Corporation, 2022), NSG (Fu et al., 2019) and SSG (Fu et al., 2021), which apply ND on the NP-based graph EFANNA. SPTAG (Chen et al., 2018) combined DC with ND.

Vamana (Subramanya et al., 2019) 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 (Croft et al., 2021; Yahoo Japan Corporation, 2022), but we classify them as ND-based per the ideas proposed in the original papers. HCNNG (Munoz et al., 2019) was influenced by SPTAG and adopted a DC approach for constructing the graph without adopting ND. ELPIS (Azizi et al., 2023) also adopted a DC strategy but leveraged both II and ND. HVS (Lu et al., 2021) and LSHAPG (Zhao et al., 2023) both propose new seed selection structures for HNSW, with the latter addition-

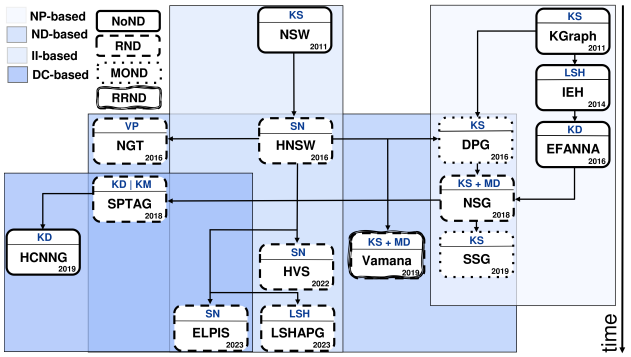


Figure 2. Graph-based ANN indexing paradigms

ally 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 3).

3. Experimental Evaluation

We experimentally evaluate twelve state-of-the-art graph-based vector search methods, based on the two key paradigms, namely seed selection and neighborhood diversification. 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 (url, 2025).

3.1. Framework

Setup. All methods were compiled with GCC 8.2 on Ubuntu 20.04 and run on a 4-socket Intel Xeon Platinum 8276 server (28 cores/socket, 1 thread/core) with 1.5TB RAM.

Algorithms. We cover the following methods: HNSW (Malkov & Yashunin, 2020), NSG (Fu et al., 2019), Vamana (Subramanya et al., 2019), DPG (Li et al., 2019), EFANNA (Fu & Cai, 2016), HCNNG (Munoz et al., 2019), KGraph (Dong, 2022), NGT (Yahoo Japan Corporation, 2022), DPG (Li et al., 2019), and two versions of SPTAG (Chen et al., 2018) (SPTAG-BKT and SPTAG-KDT, using BKT and K-D Trees, respectively). We also include ELPIS (Azizi et al., 2023) and LSHAPG (Zhao et al., 2023). IEH and FANNG are excluded due to suboptimal performance (Wang et al., 2021; Fu et al., 2019), and HVS due to difficulties running the official implementation (Lu, 2023). Euclidean distance is employed as the sole similarity metric throughout all methods and experiments. We use the most efficient publicly available C/C++ implementations for each algorithm, leveraging multithreading and SIMD vectorization to optimize performance. We also disabled the optimizations that would lead to an unfair evaluation such as cache pre-warming and L2-normalized Euclidean distance. Since all methods except ELPIS and HNSW use a single priority queue, we modified their original two-queue implementations (url, 2022; 2019) to use a single queue. The modified versions are documented in (url, 2025).

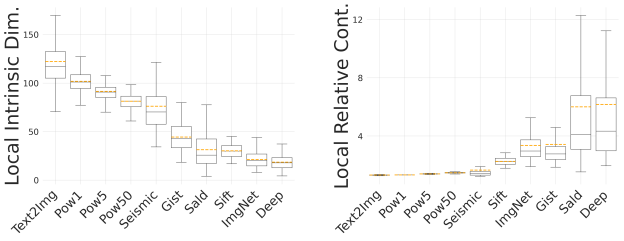
Datasets. We use seven real-world datasets from various domains, including deep network embeddings, computer vision, neuroscience, and seismology: (i) *Deep* (Skoltech Computer Vision, 2018) contains 1 billion 96-dimensional vectors extracted from the final layers of a convolutional neural network; (ii) *Sift* (TEXMEX Research Team, 2018) consists of 1 billion 128-dimensional SIFT vectors representing image feature descriptors; (iii) *Sald* (University, 2018) provides neuroscience MRI data with 200 million 128-dimensional data series; (iv) *Seismic* (for Seismology with Artificial Intelligence, 2018) contains 100 million 256-dimensional time series representing earthquake recordings from seismic stations worldwide; (v) *Text-to-Image* (Baranchuk & Babenko, 2021) 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* (Jégou et al., 2011) contains 1 million 960-dimensional vectors, using GIST descriptors (Oliva & Torralba, 2001) to capture spatial structure and color layout of images; and (vii) *ImageNet1M*, a new dataset that we generated from the original ImageNet (Russakovsky et al., 2015), producing embeddings of 1 million original vectors using the ResNet50 model (He et al., 2016), 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 test robustness across distributions, we generate three 25GB synthetic datasets (RandPow0, RandPow5, RandPow50; 256 dimensions) using power law exponents 0 (uniform), 5, and 50 (high skew). Power law follows $Y = kX^a$, with skewness increasing with exponent a.

Dataset Complexity. We measure dataset complexity using Local Intrinsic Dimensionality (LID) and Local Relative Contrast (LRC) (He et al., 2012; Aumüller & Ceccarello, 2021), defined respectively as:

$$\mathrm{LID}(x) = -\left(\frac{1}{k}\sum_{i=1}^{k}\log\frac{\mathrm{dist}_i(x)}{\mathrm{dist}_k(x)}\right)^{-1} \quad (1)$$

$$\mathrm{LRC}(x) = \frac{\mathrm{dist}_{\mathrm{mean}}(x)}{\mathrm{dist}_k(x)} \quad (2)$$

where $\text{dist}_i(x)$ is the Euclidean distance from x to its i-th nearest neighbor, $\text{dist}_{mean}(x)$ is the average distance from x to all dataset points, and $k = 100$. Lower LID and higher LRC values indicate easier search tasks. Figure 3 (computed on 1M-point dataset samples) confirms consistency between these metrics, with orange lines indicating dataset mean values. Pow0, Pow5, Pow50, Seismic, and Text2Img have the highest LID and lowest LRC (hardest datasets), whereas Sift, Deep, and ImageNet have the lowest LID and highest LRC (easiest datasets).



(a) Local intrinsic dim. (b) Local relative contrast

Figure 3. Dataset complexity

Queries. Each query workload consists of 100 vectors processed sequentially to simulate realistic, unpredictable query streams (Palpanas & Beckmann, 2019; Gogolou et al., 2020); results for 1M queries are extrapolated from these workloads. For Deep, Sift, Gist, and Text-to-Image(Text2Img), queries are randomly sampled from their provided workloads; for Sald, ImageNet, and Seismic, queries are randomly selected from datasets and excluded from index building. Query hardness experiments use Deep dataset vectors perturbed by Gaussian noise ($\mu = 0$, variance σ^2 from 0.01 to 0.1), labeled as 1%–10% noise based on the variance (Zoumpatianos et al., 2018). Queries for power-law datasets follow the original distribution with new random seeds. Unless stated otherwise, experiments use standard 10-NN queries (Simhadri et al., 2022; Aumüller et al., 2017), except for dataset complexity and seed-selection analyses, where 100-NN queries are used for higher precision and to reflect increased overhead.

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.

3.2. Neighborhood Diversification

We compare existing ND methods with the baseline NOND on II-graph (HNSW) (Fig.4). RND consistently ranks first, slightly ahead of MOND, followed by RRND; NOND performs worst across all scales (see Appendix for full results).

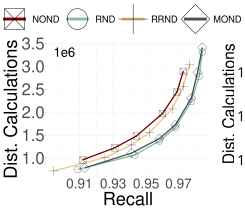
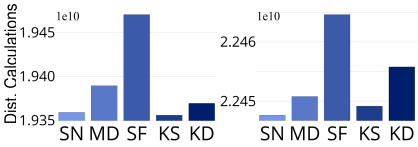


Figure 4. Impact of ND choice on query performance (Deep1B)



(a) Deep25GB (b) Deep1B

Figure 5. Impact of SS choice on query performance

3.3. Seed Selection

We evaluate seed Selection methods on II-graph with RND pruning across different datasets and scales. Both SN and KS lead in query performance. KS edges out SN on small scale (Fig.4a), while SN outperforms at large scale (Fig.4b) (see full results in appendix).

3.4. Indexing Efficiency

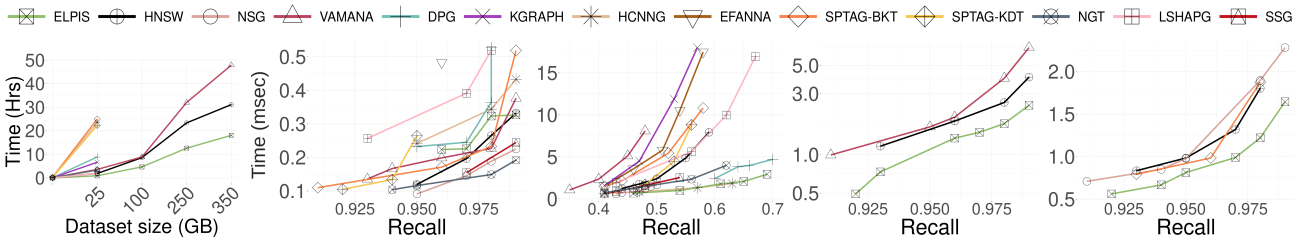
Figure 11 shows II-based methods offer the fastest indexing. ELPIS leads, being 2.7× faster than HNSW and 4× faster than NSG on 1M and 25GB. NSG is slowed by its EFANNA base, while SPTAG variants are 24× slower than ELPIS due to costly multi-tree construction. Most graph-based methods fail to scale to large collections; only HNSW, ELPIS, and Vamana index the 1B scale efficiently. ELPIS remains fastest, outperforming HNSW and Vamana by 2× and 2.7×, respectively. We also assess the memory footprint of SOTA methods; see Appendix for details.

3.5. Query Performance

We evaluate query performance across varying dataset scale and difficulties (Figure 7). On small, low-LID datasets, NSG, SSG, and HNSW perform best (Figure 6a). As dataset complexity increases—e.g., Seismic1M (Figure 6b) and Deep25GB with 10% noise (Figure 6d)—DC-based methods like SPTAG and ELPIS outperform others. Several methods, including NSG, fail to scale due to memory or indexing constraints. At billion-scale, ELPIS achieves the best performance (Figure 6c), leveraging its multithreaded design. More results across various datasets and scales are available in the Appendix.

4. 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 1 summarizes the evaluation across key criteria: for search, we assess efficiency, accuracy, and the number of tunable parameters; for indexing, we evaluate



(a) Deep1M (b) Seismic1M (c) Deep1B (d) 10% noise

Figure 6. Indexing time

Figure 7. Query performance

✓ Good ∼ Medium × Bad

Method	Query answering			Index building		
	Efficiency	Accuracy	Tuning	Efficiency	Footprint	Tuning
HNSW	✓	✓	✓	✓	✓	✓
VAMANA	✓	✓	✓	✓	✓	∼
NSG	✓	✓	✓	∼	∼	∼
SSG	✓	✓	✓	∼	∼	∼
ELPIS	✓	✓	∼	✓	✓	∼
EFANNA	×	∼	×	×	×	×
KGRAPH	×	×	×	×	×	×
DPG	×	∼	∼	∼	∼	∼
SPTAG-BKT	∼	✓	×	×	✓	×
HCNNG	✓	✓	✓	✓	✓	∼
LSHAPG	×	∼	×	∼	✓	✓
NGT	∼	∼	×	×	✓	×
SPTAG-KDT	∼	∼	∼	×	✓	×

Table 1. Comparative analysis of graph-based ANN methods.

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 parlayANN 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 10), 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. 16b, 16a, 17c, 17a, and 17b) 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 (Johnson et al., 2019) 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 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 answers, as opposed to running the search on the entire dataset, resulting in lower accuracy (Figures 16d, 17c, 17e, and 17f).

Neighborhood Diversification. Adopting an ND strategy to sparsify the graph *always* leads to better search performance, especially as dataset size grows (Figures 16d, 17c). Our experiments show that RND and MOND achieve the best performance overall (Figure 9). While RRND can mimic RND by setting $\alpha = 1$, relaxing pruning via $\alpha > 1$ allows control over edge density, which benefits disk-based and in-memory searches differently. For example, DiskANN (Subramanya et al., 2019) uses denser graphs to reduce disk I/O, trading off additional distance computations. We see promising improvements from increased density on hard datasets, but further theoretical work is needed to balance proximity and sparsity for efficient, well-connected graphs.

Seed Selection: Our experiments demonstrate that the SS strategy plays a crucial role in enhancing not only search performance (Figure 10) but also indexing efficiency (Table 3). An important research direction is to develop novel, lightweight SS strategies. Such strategies could significantly improve the overall performance of graph-based vec-

tor 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 10c, 10f).

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. 3) and hard query workloads (Fig. 20a, 16d, 17c, 19, 17f, 17e). 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.

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 ParlayANN (Manohar et al., 2024) (Fig. 21) highlight the value of optimized implementations. For example, HCNNG_Opt scaled to 1B datasets, while the non-optimized version failed beyond 25GB. Other methods could similarly benefit, underscoring the need for broader community support of libraries like ParlayANN.

Green Vector Search. Our findings highlight that choices like Neighborhood Diversification (ND) and Seed Selection (SS) reduce not only search time and memory, but also the number of distance computations and overall floating-point operations, which are key contributors to energy usage. This makes them simple yet effective levers for aligning vector search with Green AI goals. We advocate for incorporating such computation-aware metrics into future evaluations to better guide the development of efficient and sustainable search methods.

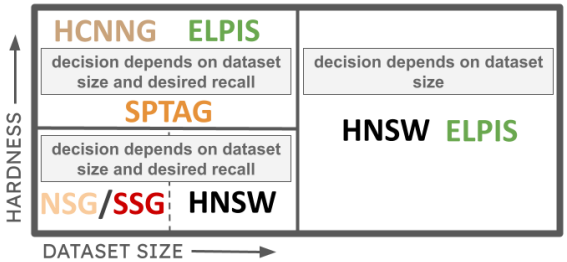


Figure 8. Recommendations (Indexing + 10K queries)

Recommendations. Our study demonstrates varying performance trends across datasets of different sizes, query workloads of different hardness and desired recall values. Figure 8 provides recommendations for methods based on these criteria. For small to medium-sized datasets (25GB and below), II-ND and ND based methods consistently demonstrate excellent performance on easier datasets (Fig.16a, 16b, 16f, 16e). On harder datasets, DC-based methods prove more efficient (Figs. 16d 17c, 16c, 17b, 19a, 19b). On large datasets (100GB and above), II-ND based methods consistently rank as top choices (Figs.18, 20).

5. 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. 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.

Acknowledgements

Supported by EU Horizon projects AI4Europe (101070000), TwinODIS (101160009), ARMADA (101168951), DataGEMS (101188416), RECITALS (101168490), $YΠAIΘA$ & NextGenerationEU project HARSH ($YΠ3TA$ – 0560901), and CNRS-UM6P program for postdoctoral fellowships.

Impact Statement

Paper presents work that advances field of Machine Learning; many potential societal consequences, none of which must be specifically highlighted here.

References

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

ELPIS Archive. https://github.com/scalablesimilaritysearch/ELPIS, 2022.

Graph-based vector search: An experimental evaluation of the state-of-the-art (archive). https://github.com/iliasazizi/GVS, 2025.

Aumüller, M. and Ceccarello, M. The role of local dimensionality measures in benchmarking nearest neighbor search. *Inf. Syst.*, 101:101807, 2021. doi: 10.1016/J.IS.2021.101807. URL https://doi.org/10.1016/j.is.2021.101807.

Aumüller, M., Bernhardsson, E., and Faithfull, A. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *International Conference on Similarity Search and Applications*, pp. 34–49. Springer, 2017.

Aurenhammer, F., Klein, R., and Lee, D.-T. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company, 2013.

Azizi, I., Echihabi, K., and Palpanas, T. Elpis: Graph-based similarity search for scalable data science. *PVLDB*, 16 (6), 2023.

Azizi, I., Echihabi, K., and Palpanas, T. Graph-based vector search: An experimental evaluation of the state-of-the-art. *Proceedings of the ACM on Management of Data*, 3(1): 1–31, 2025.

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

Baranchuk, D. and Babenko, A. Text-to-image dataset for billion-scale similarity search. https://research.yandex.com/datasets/text-to-image-dataset-for-billion-scale-similarity-search, 2021.

Beaumont, O., Kermarrec, A.-M., Marchal, L., and Rivière, É. Voronet: A scalable object network based on voronoi tessellations. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pp. 1–10. IEEE, 2007a.

Beaumont, O., Kermarrec, A.-M., and Rivière, É. Peer to peer multidimensional overlays: Approximating complex structures. In *International Conference On Principles Of Distributed Systems*, pp. 315–328. Springer, 2007b.

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

Blattmann, A., Rombach, R., Oktay, K., Müller, J., and Ommer, B. Retrieval-augmented diffusion models. *Advances in Neural Information Processing Systems*, 35, 2022.

Boniol, P. and Palpanas, T. Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series. *PVLDB*, 2020.

Bubeck, S. and von Luxburg, U. Nearest neighbor clustering: A baseline method for consistent clustering with arbitrary objective functions. *JMLR*, 10, 2009.

Camerra, A., Shieh, J., Palpanas, T., Rakthanmanon, T., and Keogh, E. Beyond One Billion Time Series: Indexing and Mining Very Large Time Series Collections With iSAX2+. *Knowledge and information systems*, 39(1): 123–151, 2014.

Chen, Q., Wang, H., Li, M., Ren, G., Li, S., Zhu, J., Li, J., Liu, C., Zhang, L., and Wang, J. *SPTAG: A library for fast approximate nearest neighbor search*, 2018. URL https://github.com/Microsoft/SPTAG.

Chiang, H.-W., Huang, C.-T., Cheng, H.-Y., Tseng, P.-H., Lee, M.-H., and Wu, A.-Y. Efficient and reliable vector similarity search using asymmetric encoding with nand-flash for many-class few-shot learning. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pp. 93–99, 2025.

Christophides, V., Efthymiou, V., Palpanas, T., Papadakis, G., and Stefanidis, K. An overview of end-to-end entity resolution for big data. *ACM Computing Surveys (CSUR)*, 53(6):1–42, 2020.

Croft, D., Gupta, M., Josifovski, V., Narayanan, P., Wu, W., and Xue, Y. Diskann: Fast approximate nearest neighbor search on disk. GitHub repository, 2021. URL https://github.com/microsoft/DiskANN. Accessed: 2024-10-25.

Dasgupta, S. and Freund, Y. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 537–546, 2008.

DBAIWangGroup. 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, 2023. GitHub repository.

Desislavov, R. et al. Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38, 2023.

Dobkin, D. P., Friedman, S. J., and Supowit, K. J. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5(4):399–407, 1990.

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

Dong, W., Moses, C., and Li, K. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pp. 577–586, 2011.

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

Echihabi, K., Zoumpatianos, K., and Palpanas, T. High-dimensional similarity search for scalable data science. ICDE, 2021.

Echihabi, K., Fatourou, P., Zoumpatianos, K., Palpanas, T., and Benbrahim, H. Hercules Against Data Series Similarity Search. *PVLDB*, 15(10), 2022.

Edelsbrunner, H. *Algorithms in Combinatorial Geometry*. Springer Publishing Company, Incorporated, 1st edition, 2012. ISBN 3642648738.

Engel, A., Monasson, R., and Hartmann, A. K. On large deviation properties of erdös–rényi random graphs. *Journal of Statistical Physics*, 117(3):387–426, 2004.

Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., and El Abbadi, A. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the ninth international conference on Information and knowledge management*, pp. 202–209, 2000.

for Seismology with Artificial Intelligence, I. R. I. Seismic Data Access. http://ds.iris.edu/data/access/, 2018.

Fortune, S. Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry*, pp. 225–265, 1995.

Fu, C. and Cai, D. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228*, 2016.

Fu, C., Xiang, C., Wang, C., and Cai, D. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, 2019.

Fu, C., Wang, C., and Cai, D. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Gabriel, K. R. and Sokal, R. R. A new statistical approach to geographic variation analysis. *Systematic zoology*, 18 (3):259–278, 1969.

Gionis, A., Indyk, P., Motwani, R., et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pp. 518–529, 1999.

Gogolou, A., Tsandilas, T., Echihabi, K., Palpanas, T., and Bezerianos, A. Data Series Progressive Similarity Search with Probabilistic Quality Guarantees. In *SIGMOD*, 2020.

He, J., Kumar, S., and Chang, S.-F. On the difficulty of nearest neighbor search. *arXiv preprint arXiv:1206.6411*, 2012.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Huang, C.-T., Chang, C.-Y., Cheng, H.-Y., and Wu, A.-Y. Bore: Energy-efficient banded vector similarity search with optimized range encoding for memory-augmented neural network. In *2024 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, 2024.

Iwasaki, M. Pruned bi-directed k-nearest neighbor graph for proximity search. In *International Conference on Similarity Search and Applications*, pp. 20–33. Springer, 2016.

Jégou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 33, pp. 117–128. IEEE, 2011.

Jin, Z., Zhang, D., Hu, Y., Lin, S., Cai, D., and He, X. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics*, 44(11): 2167–2177, 2014.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7 (3):535–547, 2019.

Kaneko, H. Local interpretation of nonlinear regression model with k-nearest neighbors. *Digital Chemical Engineering*, 6:100078, 2023.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

Kleinberg, J. et al. Small-world phenomena and the dynamics of information. *Advances in neural information processing systems*, 1:431–438, 2002.

Kleinberg, J. M. Navigation in a small world. *Nature*, 406 (6798):845–845, 2000.

Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., and Lin, X. Approximate nearest neighbor search on high dimensional data: experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2019.

Lu, K. Hvs: Hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search, 2023. URL https://github.com/Kejing-Lu/hvs.

Lu, K., Kudo, M., Xiao, C., and Ishikawa, Y. Hvs: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 15(2):246–258, 2021.

Malinen, M. I. and Fränti, P. 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*, pp. 32–41. Springer, 2014.

Malkov, Y., Ponomarenko, A., Logvinov, A., and Krylov, V. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45: 61–68, 2014.

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

Manohar, M. D., Shen, Z., Blelloch, G., Dhulipala, L., Gu, Y., Simhadri, H. V., and Sun, Y. 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*, pp. 270–285, 2024.

Matula, D. W. and Sokal, R. R. Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical analysis*, 12(3): 205–222, 1980.

Morozov, S. and Babenko, A. Non-metric similarity graphs for maximum inner product search. *Advances in Neural Information Processing Systems*, 31, 2018.

Munoz, J. V., Gonçalves, M. A., Dias, Z., and Torres, R. d. S. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition*, 96:106970, 2019.

Oliva, A. and Torralba, A. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42:145–175, 2001.

Palpanas, T. Data series management: The road to big sequence analytics. *ACM SIGMOD Record*, 44(2):47–52, 2015.

Palpanas, T. and Beckmann, V. Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA). *ACM SIGMOD Record*, 48(3), 2019.

Petitjean, F., Forestier, G., Webb, G. I., Nicholson, A. E., Chen, Y., and Keogh, E. J. Dynamic time warping averaging of time series allows faster and more accurate classification. In *ICDM*, 2014.

Ponomarenko, A., Malkov, Y., Logvinov, A., and Krylov, V. Approximate nearest neighbor search small world approach. In *International Conference on Information and Communication Technologies & Applications*, volume 17, 2011.

Pugh, W. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

Reddy, D. R. et al. Speech understanding systems: A summary of results of the five-year research effort. *Department of Computer Science. Camegie-Mell University, Pittsburgh, PA*, 17:138, 1977.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115: 211–252, 2015.

Shamos, M. I. and Hoey, D. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pp. 151–162. IEEE, 1975.

Simhadri, H. V., Williams, G., Aumüller, M., Douze, M., Babenko, A., Baranchuk, D., Chen, Q., Hosseini, L., Krishnaswamy, R., Srinivasa, G., Subramanya, S. J., and Wang, J. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. *CoRR*, abs/2205.03763, 2022. doi: 10.48550/arXiv.2205.03763. URL https://doi.org/10.48550/arXiv.2205.03763.

Skoltech Computer Vision. Deep billion-scale indexing. http://sites.skoltech.ru/compvision/noimi, 2018.

Song, L., Pan, P., Zhao, K., Yang, H., Chen, Y., Zhang, Y., Xu, Y., and Jin, R. Large-scale training system for 100-million classification at alibaba. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2909–2930, 2020.

Subramanya, S. J., Kadekodi, R., Krishaswamy, R., and Simhadri, H. V. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 13766–13776, 2019.

Sugawara, K., Kobayashi, H., and Iwasaki, M. On approximately searching for similar word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2265–2275, 2016.

Sun, Y., Wang, W., Qin, J., Zhang, Y., and Lin, X. SRS: Solving c-approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. *PVLDB*, 8(1), 2014a.

Sun, Y., Wang, W., Qin, J., Zhang, Y., and Lin, X. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proceedings of the VLDB Endowment*, 2014b.

Tao, Y., Yi, K., Sheng, C., and Kalnis, P. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 563–576, 2009.

Team, T. P. Parlayann: A deep learning library for parallel computation. https://github.com/parlayann/parlayann, 2023. Accessed: 2024-10-25.

TEXMEX Research Team. Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr/, 2018.

Toussaint, G. T. The relative neighbourhood graph of a finite planar set. *Pattern recognition*, 12(4):261–268, 1980.

Toussaint, G. T. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface*, 34, 2002.

University, S. 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, 2018.

Wang, J., Wang, N., Jia, Y., Li, J., Zeng, G., Zha, H., and Hua, X.-S. Trinary-projection trees for approximate nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 36(2):388–403, 2013.

Wang, J., Huang, P., Zhao, H., Zhang, Z., Zhao, B., and Lee, D. L. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *KDD*, 2018.

Wang, M., Xu, X., Yue, Q., and Wang, Y. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11):1964–1978, jul 2021. ISSN 2150-8097. doi: 10.14778/3476249.3476255. URL https://doi.org/10.14778/3476249.3476255.

Wang, Q., Ileana, I., and Palpanas, T. LeaFi: Data Series Indexes on Steroids with Learned Filters. *Proc. ACM Manag. Data*, 2025.

Watts, D. J. and Strogatz, S. H. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

Weber, R., Schek, H.-J., and Blott, S. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. VLDB*, pp. 194–205, 1998.

Williams, K., Li, L., Khabsa, M., Wu, J., Shih, P. C., and Giles, C. L. A web service for scholarly big data information extraction. In *ICWS*, 2014.

Yahoo Japan Corporation. Ngt: Neighborhood graph and tree for high-dimensional data. https://github.com/yahoojapan/NGT, 2022. Accessed: 2024-10-20.

Yianilos, P. N. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Soda*, volume 93, pp. 311–21, 1993.

Zhang, Y.-M., Huang, K., Geng, G., and Liu, C.-L. Fast knn graph construction with locality sensitive hashing. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 660–674. Springer, 2013.

Zhao, X., Tian, Y., Huang, K., Zheng, B., and Zhou, X. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proceedings of the VLDB Endowment*, 16(8):1979–1991, 2023.

Zoumpatianos, K., Lou, Y., Ileana, I., Palpanas, T., and Gehrke, J. Generating data series query workloads. *The VLDB Journal*, 27(6):823–846, December 2018. ISSN 1066-8888. doi: 10.1007/s00778-018-0513-x. URL https://doi.org/10.1007/s00778-018-0513-x.

A. State-of-the-Art Approaches

This section presents the graph-based approximate nearest neighbor (ANN) methods studied in our experiments, highlighting their core principles:

KGraph (Dong, 2022) 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 $O\left(n^{1.14}\right)$ (Dong et al., 2011). This refinement process, also known as NNDescent (Dong et al., 2011) (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) (Ponomarenko et al., 2011; Malkov et al., 2014) is an approximation of a Delaunay graph which guarantees the small world property (Watts & Strogatz, 1998), i.e. the number of hops L between two randomly chosen vertices grows to the logarithm of graph size n such that $L \propto Log\,(n)$. An *NSW* graph is based on the VoroNet graph (Beaumont et al., 2007a), an extention of Kleinberg's variant of Watts-Strogatz's small world model graph (Kleinberg, 2000; Kleinberg et al., 2002), 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 (Beaumont et al., 2007a). The resulting graph was proved to guarantee the small world network property (Beaumont et al., 2007a;b).

Iterative Expanding Hashing (IEH) (Jin et al., 2014) 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 (Gionis et al., 1999) and IEH-ITQ (Zhang et al., 2013). All these methods use NNDescent to finalize the graph connections.

EFANNA (Fu & Cai, 2016) selects seeds similarly to KGraph (Dong, 2022) and IEH (Jin et al., 2014) and refines candidates using NNdescent. It builds an approximate k-NNG by selecting initial neighbors of each node using randomized truncated K-D Trees (Dasgupta & Freund, 2008) and refining the graph using NNDescent (Dong et al., 2011). 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) (Malkov & Yashunin, 2020) improves the scalability of NSW (Ponomarenko et al., 2011; Malkov et al., 2014) 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) (Li et al., 2019) extends KGraph (Dong, 2022) 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 (DBAIWangGroup, 2023) utilizes RND rather than MOND for neighborhood diversification.

NGT (Yahoo Japan Corporation, 2022) 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 (Iwasaki, 2016), while the other incrementally builds graphs similar to HNSW with a range-based search strategy (Sugawara et al., 2016). In this study, we consider the former (Iwasaki, 2016). 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 (Yianilos, 1993) to select seed nodes for accurate query results.

Navigating Spreading-out Graph (NSG) (Fu et al., 2019), 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 (Chen et al., 2018) 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 (Beis & Lowe, 1997) or Balanced K-means Trees (Malinen & Fränti, 2014). 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 (Wang et al., 2013), 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 (Subramanya et al., 2019) is similar to NSG in considering the set of visited nodes when building long-range edges within the graph. However, instead of using EFANNA (Fu & Cai, 2016), Vamana uses a randomly generated graph with node degree $\geq log(n)$ to ensure the initial graph connectivity (Engel et al., 2004). 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 (Fu et al., 2021) integrates the MOND approach from DPG (Li et al., 2019) and closely follows the steps of NSG (Fu et al., 2019) and DPG (Li et al., 2019) in index building from a foundational graph. Instead of performing a search for each node to acquire candidates, SSG (Fu et al., 2021) 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 (Fu et al., 2021) enhances connectivity by constructing multiple DFS trees from various random points, in contrast to NSG's (Fu et al., 2019) singular DFS approach.

Hierarchical Clustering-based Nearest Neighbor Graph (HCNNG) (Munoz et al., 2019) 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 (Beis & Lowe, 1997), to identifying entry points during query search.

HVS (Lu et al., 2021) 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 (Zhao et al., 2023) combines HNSW graphs with multiple hash tables based on the LSB-Tree structure (Tao et al., 2009) 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 (Azizi et al., 2023) is a DC-based approach that splits the dataset into subsets using the Hercules EAPCA tree (Echihabi et al., 2022), where each leaf corresponds to a different subset, then builds in parallel a graph-based index for each leaf using HNSW (Malkov & Yashunin, 2020). 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.

B. Experimental Results

This section presents detailed experimental results across various datasets and sizes. We analyze key design choices in Neighborhood Diversification and Seed Selection, assessing their impact on query and indexing performance. We also evaluate twelve state-of-the-art methods on real and synthetic datasets of varying scale, dimensionality, and complexity. The importance of optimized implementation is highlighted. Additional details are available in the supplementary material (url, 2025).

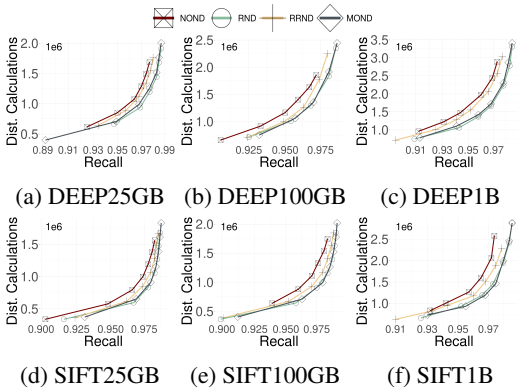
Procedure. We tune each method to achieve the best trade-offs in accuracy/efficiency. Then, we carry experiments in two steps: indexing building and query answering, with caches cleared before each step and kept warm during the same query workload. Methods were allowed at most 48 hours to build a single 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 (url, 2025).

Implementations. The implementations of various methods compared in this study were carefully examined and

sourced from their official open-source repositories. Some approaches, such as VAMANA (Subramanya et al., 2019) and NGT (Yahoo Japan Corporation, 2022), are complex and continuously evolving; for instance, VAMANA offers an improved version with incremental insertion building (Croft et al., 2021). However, we utilized the earlier version as described in the original paper, employing the two pruning steps and starting from a random graph. Ideally, all methods would be re-implemented from scratch to ensure uniformity, but this is highly time-consuming. Instead, by carefully adjusting each implementation to adhere as closely as possible to the original algorithms and applying comparable levels of optimization across methods, we provide a fair basis for comparing different algorithmic approaches for construction and search. This also enables a thorough analysis of the impact of various strategies on indexing and search performance. For comparisons involving seed selection and neighborhood diversification components, we built upon the official HNSW codebase (url, 2019), implementing additional components on top of it. All code is publicly available (url, 2025). We believe our implementation will facilitate future research by enabling reproducibility of our findings across different scenarios, as well as serving as a foundation for exploring novel approaches to neighborhood diversification and seed selection.

Comparison. We compare and evaluate different methods based on indexing time, memory footprint, and disk size, while search comparisons focus on memory usage and query time. Search time is reported for evaluating state-of-the-art (SOTA) methods. Following reviewers comments, we believe that distance calculations provide a more neutral metric for comparing different SOTA approaches, as they typically employ similar search procedures. However, methods like ELPIS leverage lower bounding distances across multi-resolution dimensionalities, which complicates the straightforward summation of distance calculations. Nonetheless, in future extensions of this work, we plan to report the number of distance calculations alongside query time for a more comprehensive evaluation of SOTA methods.

It is also important to note that this study focuses on in-memory ANN search. While our insights and results apply to scenarios where the entire dataset fits within a single machine's memory, we do not claim they extend to settings where data or graph indices must be maintained on disk or in distributed environments. We believe that techniques such as relaxed RND and K-random sampling may be preferable choices in such settings, including disk-based graph ANN. Future work could extend this study by experimenting with various graph ANN components and techniques across different settings and ANN search variants.



(a) DEEP25GB (b) DEEP100GB (c) DEEP1B

(d) SIFT25GB (e) SIFT100GB (f) SIFT1B

Figure 9. ND methods performance on real-world datasets

B.1. Neighborhood Diversification

We now evaluate the ND strategies covered in Section 2, 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 $(1.2 - 2)$ and θ $(50° - 80°)$, respectively, and selected the best values $\alpha = 1.2$ and $\theta = 60°$, which align with recommendation in (Subramanya et al., 2019; Fu et al., 2021). 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 9 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 9f, 9c). 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.

As noticed, both RND and MOND lead to the best performance, while RRND can be adjusted through α to prune similarly to RND using $\alpha = 1$. Nevertheless, controlling edge density through α and increasing edge density may deliver better performance in other settings, such as disk-based graph ANN. In particular, Vamana has been adopted in DiskANN (Subramanya et al., 2019) as the base graph to reduce the graph search diameter, thereby reducing the

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

Table 2. Pruning ratios of ND methods on Deep and Sift datasets.

number of disk I/Os, which are typically more expensive than computing extra distances per hop. We also note that increasing edge density for in-memory search on hard datasets is promising. Our experiments in the supplementary materials (url, 2025) on power-law random datasets show that RRND and MOND can deliver better performance on challenging datasets.

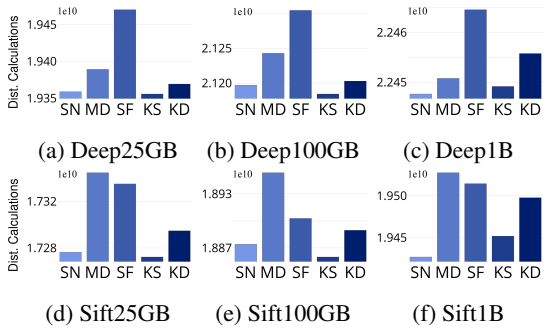
In Table 2, we report the pruning ratios of the three neighborhood ND methods on the Deep and Sift 25GB datasets. The pruning ratio quantifies the percentage reduction in the size of the candidate neighbor list before and after the diversification step. Higher pruning ratios indicate more aggressive pruning, which directly affects the graph size and memory usage. RND achieves 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.

B.2. Seed Selection

State-of-the-art graph-based vector search methods differ in their graph construction strategies but almost universally rely on beam search (Algorithm 1) for query answering. Beam search efficiently retrieves high-quality results on well-connected graphs. A critical factor affecting search efficiency is the choice of initial nodes, called *seeds*. Better seeds reduce the number of visited nodes and speed up the search. Typically, one or more entry nodes warm the beam search priority queue; when multiple seeds are used, the search begins with the seed closest to the query, keeping the others in the queue.

In these experiments, we focus on the four most common SS strategies for the beam search algorithm: SN (Malkov & Yashunin, 2020; Azizi et al., 2023), MD (Fu et al., 2019; Subramanya et al., 2019), KS (Dong, 2022; Ponomarenko et al., 2011; Li et al., 2019; Subramanya et al., 2019; Fu et al., 2021), and KD (Fu & Cai, 2016; Chen et al., 2018; Munoz et al., 2019) (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 (Malkov & Yashunin, 2020; Azizi et al., 2023), MD (Fu et al., 2019; Subramanya et al., 2019), KS (Dong, 2022; Ponomarenko et al., 2011; Li et al., 2019; Subramanya et al., 2019; Fu et al., 2021), and KD (Fu & Cai, 2016; Chen et al., 2018; Munoz et al., 2019) (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 B. 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 10. 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 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 $\sim$1M and $\sim$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 10 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 (Ponomarenko et al., 2011; Li et al., 2019; Malkov & Yashunin, 2020; Fu et al., 2019; 2021; Subramanya et al., 2019; Azizi et al., 2023; Chen et al., 2018). 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 3) 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.



(a) Deep25GB (b) Deep100GB (c) Deep1B

(d) Sift25GB (e) Sift100GB (f) Sift1B

Figure 10. The impact of SS Methods on query answering

	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 3. The impact of SS methods on Indexing Performance

B.3. 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 (url, 2025). 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 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 11 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 12 reports the memory footprint for each index, including the raw data. To perform the evaluation, we record the peak virtual memory usage during construction. 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 (url, 2019). 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 13, 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. 12) due to merging multiple MST from numerous samples generated during hierarchical clusterings. In contrast, its final index size is less than 50GB (Fig. 13).

B.4. 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, HCNNG, and EFANNA) for the sake of clarity, as their search was significantly slower than the best baselines. Full results are in (url, 2025). 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 14 indicates that Vamana, followed by ELPIS, have the lowest memory footprint during search. Even though ELPIS has a

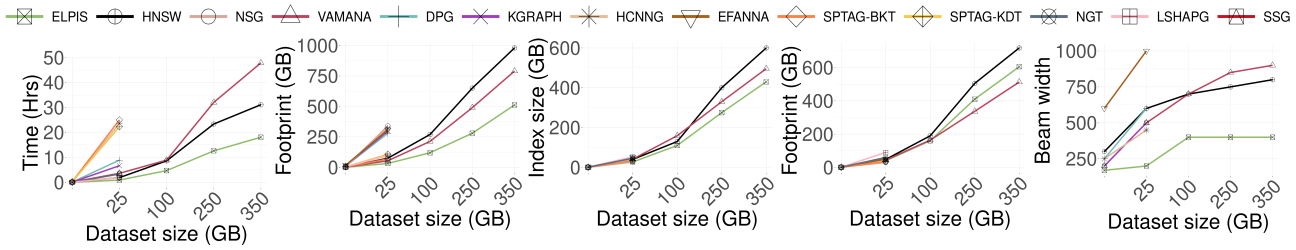


Figure 11. Indexing Time

Figure 12. Indexing Memory Footprint

Figure 13. Indexing Disk Footprint

Figure 14. Query Memory Footprint

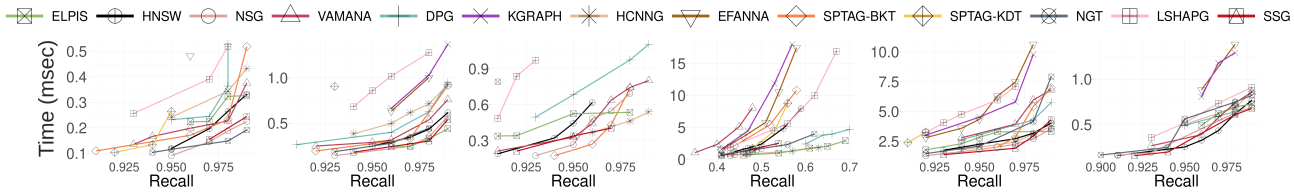
Figure 15. Query Beam Width

smaller index size, it adopts a contiguous memory storing during search just like HNSW, which increases the index footprint when loaded into memory. Besides, Figure 15 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 16), 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 17), 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 19 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 20 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 18) to 250GB (detailed results are reported in (url, 2025)).

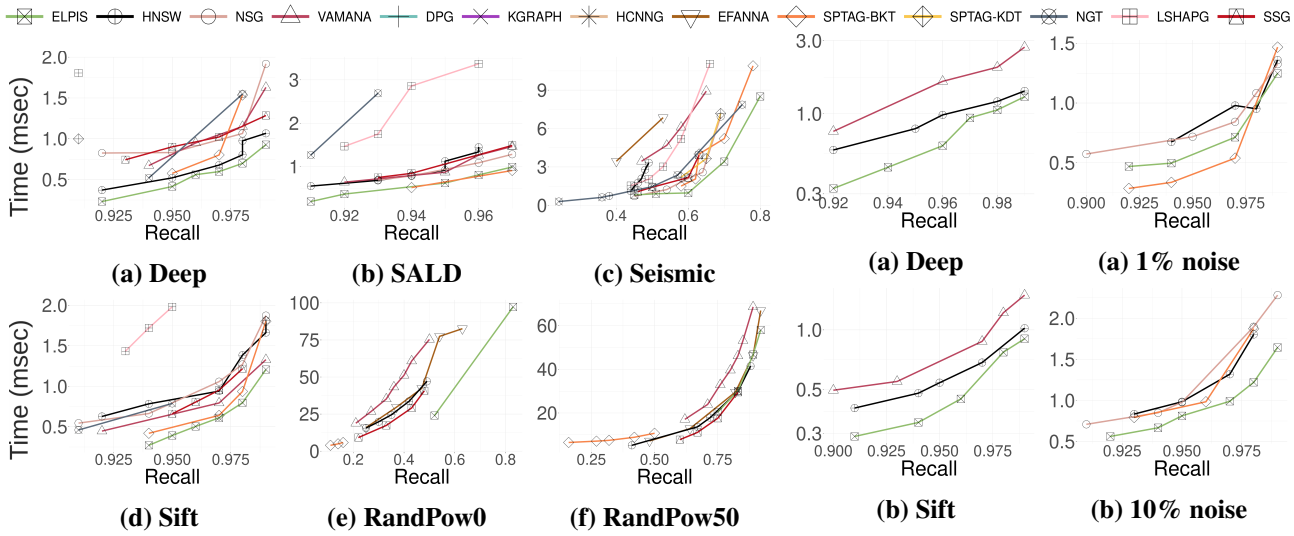
Data Distributions. We assess top performers representing different paradigms (EFANNA, Vamana, SSG, HNSW, ELPIS, and SPTAG-BKT) on challenging datasets (Fig. 3). Results (Figs. 17e and 17f) indicate that ELPIS consistently achieves high accuracy across skewness levels (0 to 50), outperforming other methods. As skewness increases, search becomes easier so most graph-based approaches improve but ELPIS maintains its superiority.

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 (Manohar et al., 2024) (Vamana_Opt, HNSW_Opt, and HCNNG_Opt) on Deep1B. Figure 21 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 (Manohar et al., 2024; Team, 2023). 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) **Deep** (b) **Sift** (c) **SALD** (d) **Seismic** (e) **Gist** (f) **Imagenet**

Figure 16. Query performance on 1M vectors



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

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

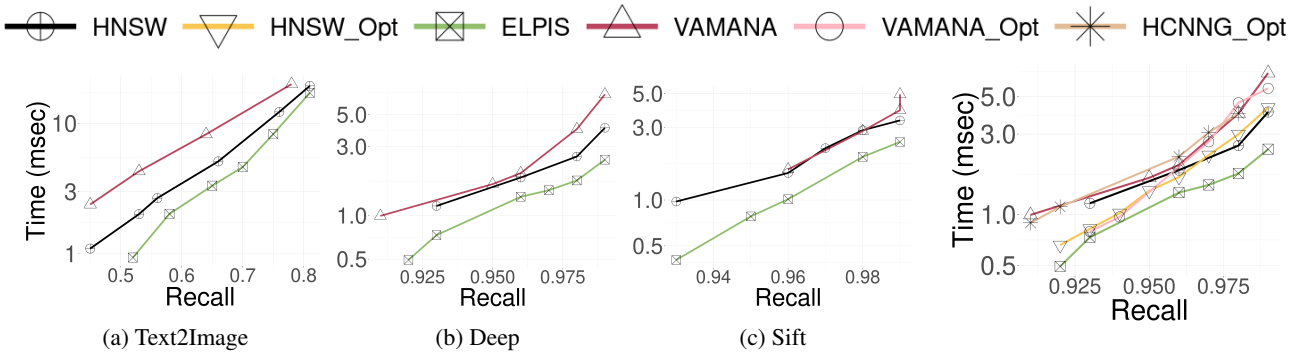
Figure 17. 25GB datasets

(a) Deep **(a) 1% noise**

(b) Sift **(b) 10% noise**

Figure 18. 100GB datasets

Figure 19. Varying workloads



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

Figure 20. 1B Datasets

Figure 21. Optimized Implementations (Deep1B)