HENN: A Hierarchical Epsilon Net Navigation Graph for Approximate Nearest Neighbor Search

Mohsen Dehghankar
Department of Computer Science
University of Illinois Chicago
Chicago, IL
mdehgh2@uic.edu

Abolfazl Asudeh
Department of Computer Science
University of Illinois Chicago
Chicago, IL
asudeh@uic.edu

Abstract

Hierarchical graph-based algorithms such as HNSW have achieved state-of-the-art performance for Approximate Nearest Neighbor (ANN) search in practice, yet they often lack theoretical guarantees on query time or recall due to their heavy use of randomized heuristic constructions. Conversely, existing theoretically grounded structures are typically difficult to implement and struggle to scale in real-world scenarios.

We propose the Hierarchical ε-Net Navigation Graph (HENN), a novel graph-based indexing structure for ANN search that combines strong theoretical guarantees with practical efficiency. Built upon the theory of ε-nets, HENN guarantees polylogarithmic worst-case query time while preserving high recall and incurring minimal implementation overhead.

Moreover, we establish a probabilistic polylogarithmic query time bound for HNSW, providing theoretical insight into its empirical success. In contrast to these prior hierarchical methods that may degrade to linear query time under adversarial data, HENN maintains provable performance independent of the input data distribution.

Empirical evaluations demonstrate that HENN achieves faster query time while maintaining competitive recall on diverse data distributions, including adversarial inputs. These results underscore HENN's effectiveness as a robust and scalable solution for fast and accurate nearest neighbor search.

1 Introduction

The Approximate Nearest Neighbor (ANN) problem involves retrieving the k closest points to a given query point q in a d-dimensional metric space. This problem is foundational in database systems, machine learning, information retrieval, and computer vision, and has seen growing importance in large language models (LLMs) [20], particularly in retrieval-augmented generation (RAG) pipelines, where relevant documents must be retrieved efficiently from large corpora [7, 25]. More generally, any vector database system implements some form of approximate nearest neighbor (ANN) search to enable efficient vector similarity queries [34, 10, 1]. For a comprehensive overview of additional applications, we refer the reader to the following surveys [40, 26, 27].

To address this problem, several classes of algorithms have been developed. Hash-based approaches (e.g., Locality-Sensitive Hashing [12, 15, 5]) offer theoretical guarantees on retrieval quality but often struggle in practical settings [36]. Quantization-based methods cluster data and search among representative centroids, yielding speedups at the cost of approximation error [18, 8, 33]. Graph-based approaches [29, 17, 40], particularly hierarchical variants [29, 28, 31], have gained attention due to

Preprint. Under review.

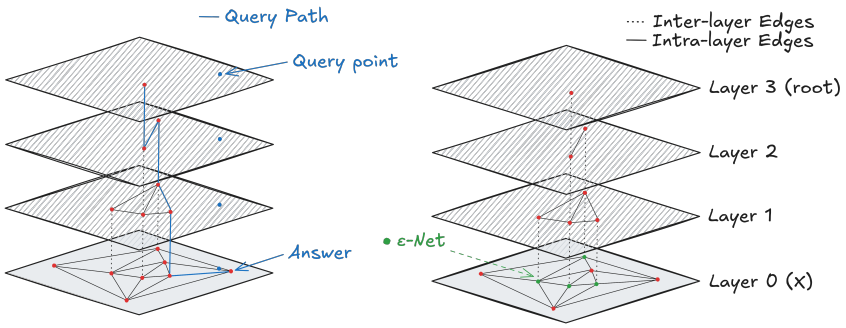


Figure 1: A simple representation of the Hierarchical ε-net Navigation Graph (right) with an example of answering a query using this structure (left). Layers are numbered bottom-up, with layer 0 being the point set X and the last layer (layer L) called the root.

their strong empirical performance and scalability. These methods build graphs over the dataset and perform greedy traversal to locate approximate neighbors quickly.[1]

Among them, the Hierarchical Navigable Small World (HNSW) [29] graph is widely used in practice. HNSW organizes data into multiple layers by assigning each point to a randomly chosen level and constructs navigable small-world graphs at each layer. While HNSW mostly delivers state-of-the-art practical performance and is easy to implement, it lacks formal guarantees on query time, and its worst-case complexity is shown to be *linear to dataset size* in adversarial settings [16, 37]. This theoretical gap limits its reliability in applications requiring performance bounds.

In contrast, earlier theoretically grounded hierarchical structures, such as Cover Trees [3] and Navigating Nets [22], provide logarithmic query time guarantees by constructing hierarchies using r-nets.[2] However, these structures are relatively *difficult to implement* and often do not scale well to real datasets, limiting their practical adoption.

This contrast highlights a gap in the literature: *no existing method offers the simplicity and scalability of HNSW while also providing provable query time guarantees* of classical hierarchical structures. Theoretical structures with strong guarantees remain underutilized due to their implementation complexity, whereas practical methods often lack worst-case guarantees.

This paper addresses this gap by proposing **Hierarchical ε-Net Navigation Graph (HENN)**, a novel graph-based structure for ANN search (Figure 1). Like HNSW, HENN constructs a hierarchical layering structure over the dataset, but its hierarchy is formed using ε-nets, a fundamental concept from computational geometry that ensures coverage of all significant neighborhoods. Each layer is built by computing an ε-net of the previous one, resulting in a well-balanced hierarchy that guarantees **polylogarithmic query time in the worst case**, depending on the specific navigation graph used in each layer. These nets can be efficiently constructed via random sampling with minimal overhead, making HENN nearly as simple to implement as HNSW.

Importantly, HENN is *modular*: any graph-based similarity search algorithm can be used within each layer, while the layer construction follows ε-net theory. This flexibility allows HENN to act as a wrapper that enhances existing graph-based solutions with theoretical guarantees depending on the chosen graph.

As the second contribution, we provide a probabilistic analysis of HNSW, showing that its hierarchy implicitly forms an ε-net with high probability, leading to a polylogarithmic query time bound under certain assumptions. This result offers theoretical insight into HNSW's strong empirical performance.

Our experimental results show that HENN achieves comparable recall and query speed to HNSW on real-world datasets, while significantly outperforming it on adversarial distributions where HNSW's performance degrades to linear time. Importantly, HENN achieves this without increasing the index size compared to HNSW. However, the improved query speed comes at the cost of increased indexing

[1] Related work is further discussed in Appendix A (supplemental material).

[2] Here, r-nets differ from ε-nets as defined in computational geometry. By r-net, we refer to a subset of points that ensures every point in the space is within distance r of some net point. In contrast, ε-nets refer to subsets that intersect all "heavy" ranges, containing more than an ε fraction of the total volume or weight.

time due to the repeated sampling required to construct ε-nets. This highlights a trade-off between indexing effort and query-time guarantees, allowing users to choose based on application constraints.

1.1 Paper Organization

The paper is organized as follows. Section 2 introduces the formal notation, defines the problem, and describes the HENN construction during the preprocessing phase, along with an overview of the query algorithm. Section 3 presents a formal analysis and proof of the query time guarantees. Section 4 reports our experimental results. Section 5 discusses the advantages and limitations of HENN. The Appendix includes extended related work, proofs, pseudo-codes, additional details on constructing ε-nets and navigation graphs, more experimental results, and a discussion of HENN in the dynamic and parallel settings.

2 Hierarchical ε-net Navigation Graph (HENN)

In this section, after formally introducing the terms and notations, we propose our data structure for answering the approximate nearest neighbor queries.

Data Model. We consider a set of n points $X = \{x_i\}^n$, where x_i is a d-dimensional point in $\mathbb{R}^d$. Additionally, we are given a distance function $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ resulting in the metric space $(X, dist)$,[3] where $dist$ measures the distance (similarity) between every pair of points. In this work, we consider any distance function resulting in a VC-dimensionality of $\Theta(d)$.[4] This is the case for most of the distance functions defined in the literature. Specifically, the ℓ_p-norm between x_i and x_j, which is measured as[5]:

$$dist(x_i, x_j) = ||x_i - x_j||_p = \left(\sum_{k=1}^{d} (x_i[k] - x_j[k])^p \right)^{\frac{1}{p}}$$

Without loss of generality, we use the Euclidean distance (ℓ_2-norm) in our examples.

Problem setting. Our objective is to preprocess X and construct a data structure that enables answering the nearest-neighbor queries. Given a query point $q \in \mathbb{R}^d$ and a point set X, a k-nearest neighbor (k-NN) query aims to find the k closest points $x_i \in X$ to q; i.e., $O_q = k\text{-}\min_{x_i \in X} dist(x_i, q)$. The Approximate Nearest Neighbor (ANN) is a relaxed version of this problem where the goal is to find a set $A_q \subset X$ of size k with a high *recall rate* – the probability that each returned point in A_q belongs to O_q, formally:

$$Recall@k = \frac{|A_q \cap O_q|}{k} \leq 1$$

We are now ready to define our data structure for ANN query answering:

Definition 1 (Hierarchical ε-net Navigation Graph (HENN)) *We define HENN as a multi-layer graph for answering ANN queries on a point set X:*

- *Each node of the graph represents a point in X.*
- *Each layer $\mathcal{L}_i$, $1 \leq i \leq L$, is an ε-net of the point-set X and its previous layer, for the range space and the values of ε specified in Section 2.1. We use $\mathcal{L}_0$ to show the point set X.*
- *The nodes within each layer are connected with* Intra-layer edges *that construct a* navigable graph *to answer the ANN* only *inside this layer. Any of the existing or future navigable-graph techniques, such as the Delaunay Triangulation (DT) [23], an approximation of the DT graph (like Navigable Small World, NSW [30]), or the k-nearest neighbor (kNN) graph, can be applied.*[6]

[3] A distance function $\mathbf{d}$ is defines a metric space if (i) $\mathbf{d}(x, x) = 0$, (ii) $\mathbf{d}(x, y) = \mathbf{d}(y, x)$, and (iii) $\mathbf{d}(x, y) + \mathbf{d}(y, z) \geq \mathbf{d}(x, z)$ (triangular inequality) [11].

[4] With a slight abuse of the term, we consider the VC-dim of a distance function $dist$ as the VC-dim of range space $(X, \mathcal{R})$, where $\mathcal{R} = \{dist(x, p) \leq r, \forall p \in \mathbb{R}^d \text{ and } r > 0\}$.

[5] Angular distances like cosine similarity also have this property. A discussion on this is also provided in the detailed experiments in Appendix G.

[6] In Appendix B.2 in the supplemental material, we propose heuristics based on ideas such as dimensionality reduction to improve the practical performance of the navigable graph used in HENN.

- *Each pair of layers $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$, $0 \leq i < L$, are connected with Inter-layer edges. there is an edge between the nodes $v \in \mathcal{L}_i$ and $u \in \mathcal{L}_{i+1}$, if and only if these v and u represent the same point in the point set X.*

Figure 1 (right) shows a simple representation of a HENN structure.

Having defined our data structure, we now present its construction details.

2.1 HENN Graph Construction

In this section, we describe the construction of the HENN graph during the preprocessing phase. We begin by outlining the high-level structure and then progressively examine each component in greater detail through separate subsections, following a top-down approach.

ε-net Hierarchy. Each layer $\mathcal{L}_i$ in the HENN graph is an ε-net for the range space $(X, \mathcal{R})$, where X is the input point set and $\mathcal{R}$ is the family of *ring* ranges, defined as follows, that specify subsets of X. For a formal definition of ε-net, please refer to the background review provided in Appendix B.1.

Definition 2 (Ring Ranges) *Given a set of points X, and a distance function $dist$, a* **ring** *$R \in \mathcal{R}$ is specified with a base point $p \in \mathbb{R}^d$ and two values $r_1 < r_2$. Any point in X with distance within two values r_1 and r_2 from p falls inside the ring. Formally,*

$$R \cap X = \{x \in X \mid r_1 \leq dist(x, p) \leq r_2\}$$

Proposition 1 *The VC-dim of the ring range space, $(X, \mathcal{R})$, is $\Theta(d)$.*

The correctness of Proposition 1 follows the fact that each ring range $R : \langle p, r_1, r_2 \rangle$ can be formulated by mixing the two distance ranges $R' : dist(x, p) \leq r_2$ and $R'' : dist(x, p) \leq r_1$ as $R = R' - R''$. Hence, due to the mixing property of range spaces [11], the VC-dim of the ring ranges is two times the VC-dim of the distance ranges, i.e., $\Theta(d)$. Remember, we assumed that the VC-dim of $dist$ is $\Theta(d)$.

Following Proposition 1, one can build an ε-net of size $O(\frac{d}{\varepsilon} \log \frac{d}{\varepsilon})$, for the ring range space, for a given value ε [12] (See ε-net construction detail background in Appendix B.1).

The construction algorithm. The preprocessing phase follows a recursive construction of the HENN graph using Algorithm 1. It begins with the initial set of points being the entire point set, i.e., $\mathcal{L}_0 = X$, and constructs an $\varepsilon(n)$-net over X, as explained in Appendix B.1, where $n = |X|$ and $\varepsilon(n)$ is defined later in Equation 1. This forms the first layer, denoted $\mathcal{L}_1$. After finding the points in this layer, we follow a black-box approach for constructing a *navigable graph* within this layer and add the intra-layer edges accordingly (see Appendix B.2 for more details). Even though this construction follows a sequential order in building layers, a discussion on how to parallelize this step is provided in Appendix D.

Algorithm 1 HENN Graph Construction (Preprocess) Algorithm

Require: The set of points X, maximum number of layer L, and the exponential decay m.
Ensure: The HENN graph $\mathcal{H}$.

```
function BUILDHENN(X, L, m)
    L_0 ← X
    for i ≤ L do
        s ← |L_{i-1}|
        ε ← calculate ε(s) based on Equation 1
        L_i ← BuildEpsNet(L_{i-1}, ε)          ▷ build an ε-net, see Appendix B.1.
        Connect each node in L_i to the previous layer (Inter-layer edges).
        Build a navigable graph on L_i (Intra-layer edges).   ▷ See Appendix B.2.
    Return H = {L_0, L_1, ··· , L_L} and the edges.
```

Subsequently, the algorithm recursively builds each layer $\mathcal{L}_{i+1}$ as an $\varepsilon(|\mathcal{L}_i|)$-net of the previous layer $\mathcal{L}_i$ and adds the inter-layer edges. This process continues until a total of L layers are constructed, where L is a hyperparameter specifying the depth of the HENN graph. The parameter ε is defined as

$$\varepsilon(s) = c_0 d \frac{\log s}{s} 2^m, \tag{1}$$

where c_0 is a sufficiently large constant (to be specified later), d is the dimension of X, and m is a hyperparameter controlling the exponential decay in the size of the layers across the hierarchy. As the following lemma shows, increasing m accelerates the exponential decay in the layer sizes.

Lemma 2 *In the construction described above, by using the equation 1, the size of each layer $\mathcal{L}_i$ would decrease exponentially. Formally, for every $1 \leq i \leq L$ ($\mathcal{L}_0 = X$),*

$$|\mathcal{L}_i| \leq \frac{|\mathcal{L}_{i-1}|}{2^m}$$

Proof *Proof is provided in Appendix E.* □

Note: The value of m (layers exponential decay), in HNSW, is generally selected between 1 and 6, based on the tradeoff between the recall and indexing time [29].

Corollary 1 *The property of being an ε-net is preserved under the addition of extra points. Therefore, without loss of generality, we may assume that for all i, the sizes of the layers satisfy:*

$$|\mathcal{L}_i| = \frac{|\mathcal{L}_{i-1}|}{2^m}$$

Corollary 2 *The largest possible number of layers, L, in the HENN graph grows logarithmically with the size of the point set X, where $|X| = n$. Specifically, we have*

$$L = O\left(\frac{\log n}{m}\right).$$

After explaining the construction of the HENN graph during the preprocessing time, we are now ready to describe ANN query answering.

2.2 Query Answering Using HENN

During the query time, given a query point $q \in \mathbb{R}^d$, the goal is to find the approximate nearest neighbor of q within X. We follow the Greedy Search algorithm on top of HENN graph: Starting from a random node in the root (layer $\mathcal{L}_L$), we find the nearest neighbor of q within this layer by following a simple Greedy Search algorithm. After finding the nearest neighbor v_i in layer $\mathcal{L}_i$, we continue this process, starting from v_i in layer $\mathcal{L}_{i-1}$. We continue this until reaching the bottom (layer $\mathcal{L}_0$).[7] A pseudo-code of this algorithm is provided in Algorithm 4 in the Appendix, and a visual illustration is shown in Figure 1 (left). Answering k-Nearest Neighbors for $k > 1$ can be achieved by considering a set of candidates at each step in the greedy algorithm [29].

3 Theoretical Analysis

In this section, we provide theoretical guarantees on the running time of the HENN graph for answering ANN. The query time for ANN using HENN partially depends on the specific navigation graph used (Appendix B.2), as in HNSW. Recall that a navigable graph connects the nodes in each layer $\mathcal{L} \subseteq X$ of the HENN graph using the intra-layer edges, and is used by the Greedy algorithm at the query time.

Let $\mathcal{G}_{\mathcal{L}} = G(\mathcal{L}, E_{\mathcal{L}})$ be a navigable graph built for a layer $\mathcal{L} \subseteq X$, where $E_{\mathcal{L}}$ is the set of intra-layer edges. Navigating through the graph $\mathcal{G}_{\mathcal{L}}$, while following the edges $E_{\mathcal{L}}$ greedily, the Greedy algorithm finds a point $\overline{p}$ as the approximate nearest neighbor of q among the nodes in $\mathcal{L}$. Formally,

$$\overline{p} = \mathsf{GS}(\mathcal{G}_{\mathcal{L}}, q)$$

Where GS is short for *GreedySearch* in Algorithm 4. In the following, we define a notation of accuracy for a navigable graph algorithm $\mathcal{G}$, which is used as a parameter in the subsequent guarantees that we provide.

[7] Note that this is the same greedy algorithm used in the literature.

Definition 3 (Recall Bound: ρ_δ) *For a given Navigable Graph $\mathcal{G}_\mathcal{L}$ on a subset $\mathcal{L}$, the Recall Bound of $\mathcal{G}_\mathcal{L}$ with a probability δ is defined as the smallest value k, such that for all queries q, the result of greedy search on $\mathcal{G}_\mathcal{L}$ will give at least one point within the k-nearest Neighbors of q with probability more than δ. In other words,*

$$\rho_\delta = \arg\min_{k \le |\mathcal{L}|} \Big(Pr\Big(\mathsf{GS}(\mathcal{G}_\mathcal{L}, q) \in \mathsf{NN}_{k,\mathcal{L}}(q) \Big) \ge \delta \Big)$$

Where $\mathsf{NN}_{k,\mathcal{L}}(q)$ is the ground-truth k-nearest neighbors of q in $\mathcal{L}$. The probability is taken over all possible initial nodes of the graph $\mathcal{G}_\mathcal{L}$, as a starting node.

Note: This defines a different and weaker notion of accuracy for graph-based algorithms compared to the standard recall rate. While $Recall@k$ measures the fraction of the true k nearest neighbors retrieved, here we are interested in identifying the smallest value of k such that, with high probability, at least one of the true k nearest neighbors is returned by the algorithm.

In practice, we are usually concerned with a fixed probability $\delta \ge 0.9$. In addition, for most of the existing navigable graphs [30, 29], $\rho_{0.9} = O(1)$. We will compare these values by running additional experiments in the Appendix.

3.1 Running Time

We start by proving a lemma that focuses only on a *single pair of layers* (the bottom-most layer and the one above it) and shows the usefulness of using ε-nets in this hierarchy.

Lemma 3 *Let $\mathcal{L}$ be an ε-net of X, where $n = |X|$. Let $\overline{p} = \mathsf{GS}(\mathcal{G}_\mathcal{L}, q)$ for a query point q. Then, the number of points in X closer than $\overline{p}$ to q is less than $\varepsilon \cdot (\rho_\delta + 2) \cdot n$, with probability more than δ. Formally, with probability more than δ, we have:*

$$|\mathsf{NN}_{\le dist(\overline{p},q)}| \le \varepsilon \cdot (\rho_\delta + 2) \cdot n \tag{2}$$

where $\mathsf{NN}_{\le dist(\overline{p},q)} = \{p \in X \mid dist(p,q) \le dist(\overline{p},q)\}$

Proof *Sketch: This is a result of finding ε-nets on ring ranges, where it bounds the total number of points around q. The proof is provided in Appendix E.* □

Now, we are ready to propose the main theorem on the time complexity of ANN-query answering using HENN graph:

Theorem 4 *Given a set of points $X \subset \mathbb{R}^d$ and the hyperparameter m for the exponential decay, let $\mathcal{H}$ be the HENN graph built using the function* BuildHENN *(Algorithm 1). Then, the running time of* $\mathsf{Query}(\mathcal{H}, q)$ *(Algorithm 4) is $O(d \cdot d^* \cdot \rho_\delta \cdot \log^2 n)$. Where d^* is the average degree of a node in the navigation graphs $\mathcal{G}$ and ρ_δ is the Recall Bound of the navigable graphs.*

Proof *Sketch: This can be proved by applying Lemma 3 inductively, considering layers $\mathcal{L}_1$ up to $\mathcal{L}_L$. The proof is provided in Appendix E* □

In most of the navigation graphs used in practice, the degree d^* is fixed as a constant. As a result, the running time is *polylogarithmic in* n in these settings. The running time also depends on the Recall Bound of the navigation graph $\mathcal{G}$ used in the HENN construction. The better navigation graphs in terms of accuracy also result in faster query answering. As long as ρ_δ is a constant (or $O(\log n)$), the running time will still be *polylogarithmic in* n.

Corollary 3 (HNSW Guarantees) *Since a random sample of a point set forms an ε-net with a certain probability, Theorem 4 implies that HNSW, which constructs each layer through random sampling, achieves the same polylogarithmic query time, assuming that each layer forms an ε-net with a specific probability. Note that HENN improves on this by explicitly enforcing the ε-net property deterministically (depending on the Recall Bound).*

3.2 Preprocessing and Indexing

The index size of HENN is exactly similar to HNSW, which is linear in n. Following the Corollaries 1 and 2, this size can be calculated as:$\sum_{i=0}^{L} |\mathcal{L}_i| = \sum_{i=0}^{L} \frac{n}{2^m} = O(n)$.

The preprocessing time for constructing the HENN index depends on the chosen method for building the ε-net. When using random sampling, a Monte Carlo procedure is employed, repeating the sampling a constant number of times until an ε-net is obtained (see Appendix B.1 for details).

4 Experiments

In this section, we evaluate the performance of the HENN structure on both synthetic (challenging) data instances and widely used approximate nearest neighbor (ANN) benchmark datasets. This section includes a comparative analysis between HENN and the HNSW structure. Additional experiments, covering a broader range of benchmark datasets and comparisons with alternative navigable graph baselines (beyond NSW), are provided in the supplemental material (Appendix G). Our code is built upon the widely-used standard C++ implementation of HNSW, hnswlib.[8] The source code is available in this repository.

4.1 Experiments Setup

The experimental environment configuration is described in Appendix G.1.

Implementation Details. The core step in constructing the HENN index involves computing an ε-net for each layer, beginning with the base layer $\mathcal{L}_0$, which includes the entire dataset. Appendix G.2 provides a detailed explanation of this process, along with the heuristics used in our implementation.

Methods and Datasets. We compare HENN against the HNSW baseline, highlighting their key difference in layer construction: HENN enforces ε-net layers, while HNSW uses random sampling. Both use navigable graphs built via greedy insertion. Experiments are conducted on both synthetic and real benchmark datasets, including SIFT [18], GloVe [35], and MNIST [24]. Additional implementation details, datasets details, and results are provided in the Appendix G.3.

Evaluation. To evaluate the methods, particularly in terms of runtime, we focus on worst-case performance. Specifically, for each reported value, we execute the method multiple times and return the maximum runtime observed, rather than the average.[9]

4.2 Experiment Results

We begin by presenting experiments on standard benchmark datasets, demonstrating that HENN achieves comparable recall and query time to the baseline methods. In the subsequent subsection, we turn to more challenging synthetic datasets with controlled skewness and non-uniformity, where HENN consistently outperforms the baselines.

4.2.1 Benchmark Datasets

In this subsection, we compare the methods on three real datasets: SIFT, GloVe, and MNIST. Other benchmark comparisons are included in the supplemental material.

Query Speed vs. Recall tradeoff. Figure 2 presents the query speed versus recall@10 for different datasets. Query speed is measured in queries per second (QPS), defined as the inverse of query time. Higher values of both recall and speed (toward the top-right of the plot) indicate a more effective ANN algorithm. On the SIFT dataset, HENN achieves performance comparable to the baseline in terms of both speed and recall. On the GloVe and MNIST datasets, HENN attains a higher area under the curve (AUC), indicating improved overall efficiency.

These results show that HENN maintains the effectiveness of prior methods on standard benchmarks, where ANN search is relatively less challenging in terms of worst-case behavior, where the difficulty primarily stems from high dimensionality and large data sizes. This also indicates that the SIFT dataset is fairly uniformly distributed and not skewed enough to challenge the baseline compared to the GloVe and MNIST.

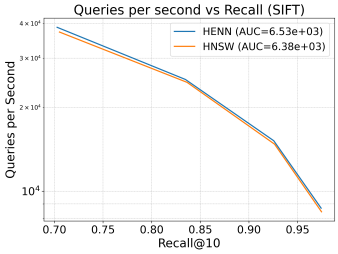
4.2.2 Synthetic Dataset

As previously discussed, we construct a non-uniform, skewed synthetic dataset with varying parameters to compare the worst-case performance of the methods.

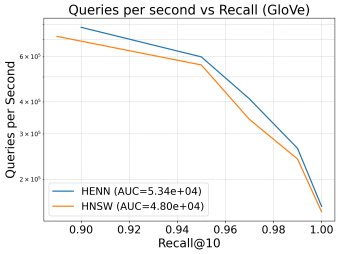
Effect of n. Figures 3 and 4 present a runtime comparison between HENN and HNSW as the dataset size n varies. Notably, HENN achieves up to a $1.7\times$ speedup over the baseline HNSW, with

[8] https://github.com/nmslib/hnswlib

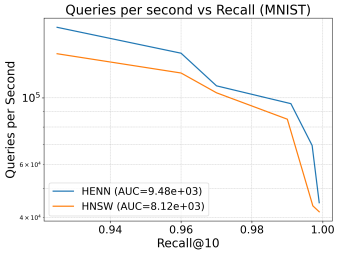
[9] Because in the average case, HNSW will behave like HENN based on Corollary 3 and Theorem 4.



SIFT: $d = 128$



GloVe: $d = 100$



MNIST: $d = 784$

Figure 2: Query speed vs. Recall@10 - up and to the right is better (higher AUC). HENN performs at least as well as the HNSW baseline on standard benchmark datasets. These plots are generated by varying the hyperparameter ef in hnswlib.

a larger performance gap as the dataset size increases. Additionally, the runtime of HENN shows a polylogarithmic growth trend, consistent with the theoretical guarantee established in Theorem 4.

Effect of d. Figure 5 presents a runtime comparison as the dimensionality d varies. Similar to previous results, HENN maintains a consistent speedup. Additionally, the runtime of HENN exhibits a linear growth with respect to the dimensionality d.

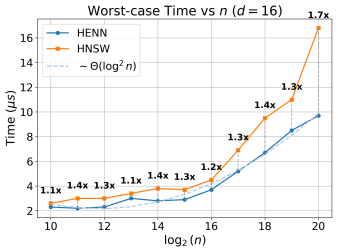


Figure 3: $d = 16$

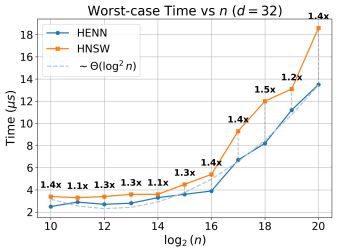


Figure 4: $d = 32$

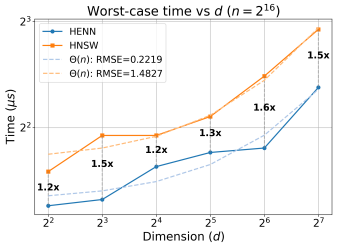


Figure 5: $n = 2^{16}$

Comparing the worst-case time of the methods. The worst-case time is calculated by running each method multiple times and using the maximum value as the aggregation. The logarithmic trend is shown with a dotted line. RMSE, root mean squared error, measures how well the trends are fitted.

Number of Hops. Figures 6 and 7 compare the two methods in terms of the total number of graph hops visited during the query execution. This metric serves as a proxy for the number of greedy steps performed by Algorithm 4 to locate the approximate nearest neighbor. HENN consistently required fewer hops, leading to fewer distance computations overall. The improvement is as great as $1.9\times$ fewer hops for HENN. This behavior reflects the more uniform and structured construction of hierarchical layers in HENN compared to the baseline. Figure 8 shows this comparison on different dimensions d.

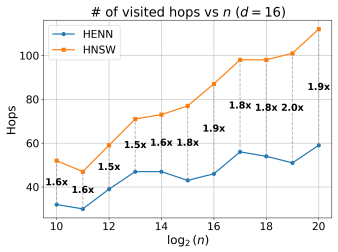


Figure 6: $d = 16$

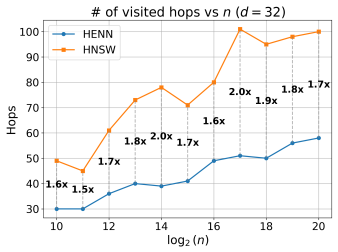


Figure 7: $d = 32$

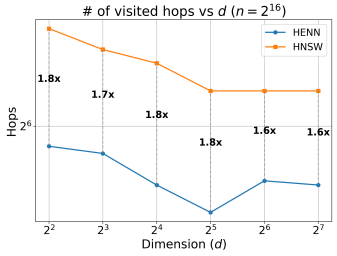
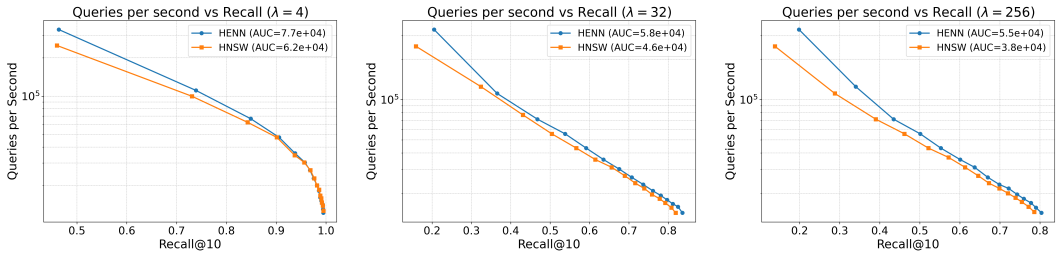


Figure 8: $n = 2^{16}$

Comparing the number of visited hops during the query time. The larger number of visited hops results in longer and less efficient query times.

Query Speed vs. Recall tradeoff. Figure 9 shows the trade-off between query speed and recall@10 for the evaluated methods. We vary the parameter λ to control the skewness of the data distribution, with larger λ producing more non-uniform datasets. Under highly skewed conditions, HENN

demonstrates more speedup over HNSW. In more uniform settings, HENN achieves comparable recall to HNSW while maintaining competitive query speed.



$\lambda = 4$ $\lambda = 32$ $\lambda = 256$

Figure 9: Query speed vs Recall@10 - up and to the right is better (higher AUC). The input is an exponential distribution. The larger the λ, the more skewed the data. In this setting, $d = 32$ and $n = 50000$.

Indexing Phase. Table 1 compares the index sizes across datasets. Due to the construction strategy of HENN, there is no significant increase in index size relative to the baseline, as the layer sizes remain unchanged. Indexing time results are shown in Table 2, where we observe an increase in construction time for HENN. This overhead arises from the repeated random sampling required to satisfy the ε-net condition. A detailed discussion of the trade-off between the success probability of finding an ε-net and indexing time is provided in the Appendix G.

Method	SIFT	GloVe	Synthetic
HENN	633.89	26.35	18.00
HNSW	633.88	26.35	18.00

Table 1: Index size (MB). The Synthetic data has $n = 2^{16}$, $d = 32$

Method	SIFT	GloVe	Synthetic
HENN	79.60	1.36	1.01
HNSW	20.28	0.24	0.29

Table 2: Indexing time (s). The Synthetic data has $n = 2^{16}$, $d = 32$

5 Advantages and Limitations

The HENN structure is easy to implement and closely resembles HNSW, with a key difference in how layers are constructed. It ensures that each layer forms an ε-net, which can be done by random sampling. HENN is also modular with respect to the choice of navigable graph used at each layer. Any graph structure suitable for similarity search, such as k-NN, Delaunay Triangulation (DT), or Navigable Small World (NSW), can be integrated into HENN.

HENN supports parallelization during the indexing phase, enabling faster construction on large datasets. It also accommodates dynamic updates, both insertions and deletions, while preserving theoretical guarantees. Further details are provided in Appendices D and C.

The HENN structure is applicable to any metric space with induced bounded VC-dimension of $\Theta(d)$ (like all widely-used ℓ_p-norm and angular distances for ANN). However, a limitation is that the theoretical guarantees no longer hold when the VC-dimension is unbounded.

The provided time guarantees of HENN depend on the properties of the used navigable graphs, such as the degree and recall bounds. In practice, commonly used graph structures often have constant degree and recall bounds. However, some graphs, such as Delaunay Triangulation (DT), can have degrees that grow exponentially with the dimension. As a result, this can cause a limitation for this structure. These theoretical guarantees come at the cost of increased indexing time. The construction time grows with the number of sampling iterations required to ensure that each layer forms an ε-net.

6 Conclusion

We introduced HENN, a hierarchical graph-based structure for ANN search that combines theoretical guarantees with practical efficiency. By constructing ε-net-based layers, HENN achieves provable polylogarithmic query time while remaining simple to implement. We also provide a probabilistic analysis for HNSW, offering insight into its empirical success. Experiments show that HENN performs robustly across both standard and adversarial datasets, making it a practical and scalable solution for nearest neighbor search.

References

[1] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104, 2006.

[4] Bernard Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, Cambridge, UK, 2000.

[5] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[6] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 3rd edition, 2008.

[7] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6491–6501, 2024.

[8] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):744–755, 2013.

[9] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. Fast incremental maintenance of approximate histograms. *ACM Transactions on Database Systems (TODS)*, 27(3):261–298, 2002.

[10] Yikun Han, Chunjiang Liu, and Pengfei Wang. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*, 2023.

[11] Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2011.

[12] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. 2012.

[13] David Haussler and Emo Welzl. Epsilon-nets and simplex range queries. In *Proceedings of the second annual symposium on Computational geometry*, pages 61–71, 1986.

[14] John Healy and Leland McInnes. Uniform manifold approximation and projection. *Nature Reviews Methods Primers*, 4(1):82, 2024.

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

[16] Piotr Indyk and Haike Xu. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations. *Advances in Neural Information Processing Systems*, 36:66239–66256, 2023.

[17] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems*, 32, 2019.

[18] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[19] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, volume 33, pages 117–128. IEEE, 2011.

[20] Zhi Jing, Yongye Su, Yikun Han, Bo Yuan, Haiyun Xu, Chunjiang Liu, Kehai Chen, and Min Zhang. When large language models meet vector databases: A survey. *arXiv preprint arXiv:2402.01763*, 2024.

[21] William B Johnson, Joram Lindenstrauss, et al. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[22] Robert Krauthgamer and James R Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Citeseer, 2004.

[23] Gérard Le Caer and Renaud Delannay. Correlations in topological models of 2d random cellular structures. *Journal of Physics A: Mathematical and General*, 26(16):3931, 1993.

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

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

[27] Yingfan Liu, Cheng Hong, and Jiangtao Jiang. Revisiting k-nearest neighbor graph construction on high-dimensional data: Experiments and analyses. *arXiv preprint arXiv:2112.02234*, 2021.

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

[29] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

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

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

[32] Stephen M Omohundro. Five balltree construction algorithms. 1989.

[33] Ezgi Can Ozan, Serkan Kiranyaz, and Moncef Gabbouj. Competitive quantization for approximate nearest neighbor search. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2884–2894, 2016.

[34] James Jie Pan, Jianguo Wang, and Guoliang Li. Survey of vector database management systems. *The VLDB Journal*, 33(5):1591–1615, 2024.

[35] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014.

[36] Ninh Pham and Tao Liu. Falconn++: A locality-sensitive filtering approach for approximate nearest neighbor search. *Advances in Neural Information Processing Systems*, 35:31186–31198, 2022.

[37] Liudmila Prokhorenkova and Aleksandr Shekhovtsov. Graph-based nearest neighbor search: From practice to theory. In *International Conference on Machine Learning*, pages 7803–7813. PMLR, 2020.

[38] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[39] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

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

Appendix

Table of Content

A Related Work

In this section, we review the literature most relevant to our work. A broader overview of techniques for the Approximate Nearest Neighbor (ANN) problem, including references to comprehensive surveys, is presented in the Introduction section. Here, we focus primarily on *hierarchical* approaches developed for solving the ANN problem.

A notable class of approaches for solving the ANN problem is based on hierarchical structures. One of the most widely used methods in this category is Hierarchical Navigable Small World (**HNSW**) [29], which constructs a multi-layered structure of navigable small-world graphs to enable efficient search. Interestingly, the core idea of hierarchical organization can be traced back to earlier work in computational geometry, including **Cover Trees** [3] and **Navigating Nets** [22].

HNSW. Hierarchical Navigable Small World (HNSW) graphs [29] construct a multi-layered hierarchy of navigable small world (NSW) graphs. An NSW graph serves as an efficient approximation of the Delaunay graph [11], which is known to be an optimal structure for solving the approximate nearest neighbor (ANN) problem [30]. The Delaunay graph is closely related to the Voronoi diagram, which partitions the space into cells based on their proximity to the points in the dataset. Unlike the Delaunay graph, which requires explicit geometric computation, NSW graphs are built incrementally by inserting points and connecting them to their approximate nearest neighbors [30]. During insertion and querying, HNSW employs a greedy search algorithm (Algorithm 4) to navigate through the graph and locate nearby points. To improve scalability, HNSW organizes the data in a hierarchy where the number of nodes decreases exponentially across layers, resulting in $O(\log n)$ layers and a total space complexity of $O(n)$.

Cover Trees and Navigating Nets. The Cover Tree data structure [3] provides a scalable and theoretically grounded approach to nearest neighbor (NN) search in general metric spaces. The cover tree maintains logarithmic query time and linear space complexity under a measure of intrinsic data dimensionality. The structure recursively organizes points into a nested hierarchy of layers, enforcing both covering and separation invariants, which enable efficient traversal and pruning during NN queries, inspired by Navigating Nets. Navigating Nets [22] is another hierarchical structure based on r-nets designed for efficient proximity search in general metric spaces. The approach relies on constructing a sequence of nested r-nets, which progressively approximate the dataset at multiple scales. r-net is a subset of points that cover an r distance around any points in the dataset. These hierarchical nets enable fast navigation by guiding the search from coarse to fine resolutions. It achieves logarithmic query and update times with respect to the number of points. Unlike randomized methods, Navigating Nets gives predictable performance. However, they are difficult to implement and do not scale well to large datasets. Because of this, they are rarely used in practical nearest neighbor applications today.

General Solutions to Nearest Neighbor Search. Solutions to the nearest neighbor problem can be categorized along several dimensions. One common distinction is between classical methods, which primarily target the exact NN problem. Examples include k-d trees [2], ball trees [32], and Delaunay triangulations [12]. While effective in low-dimensional spaces, these methods typically fail to scale in high-dimensional settings. Another broad category includes quantization-based methods [18, 8, 33], which cluster the data and represent points by their assigned centroids (codewords), thereby

approximating distances efficiently. Additionally, hashing-based methods such as Locality-Sensitive Hashing (LSH [5]) provide theoretical guarantees and have been widely used for high-dimensional ANN, and finally, the graph-based methods [30, 29, 37, 28]. For comprehensive overviews of these and other approaches, we refer the reader to the following surveys [34, 10, 1, 40, 26, 27].

B ε-net and Navigable Graphs

In this section, we provide background on the theory of ε-nets, covering both sampling-based and discrepancy-based algorithms for constructing them. We also describe the heuristics used in our experiments for building ε-nets. Finally, we discuss the background on navigation graphs that can be used in the HENN data structure.

B.1 ε-nets

First, we define ε-net for a general range system $(X, \mathcal{R})$:

Definition 4 (ε-net) *Given a range space defined as $(X, \mathcal{R})$, where X is a set of points and $\mathcal{R}$ is a family of subsets (ranges) over X, an ε-net for this system is a subset $\mathcal{A} \subseteq X$ such that:*

$$\forall R \in \mathcal{R}: \quad \frac{|X \cap R|}{|X|} > \varepsilon \;\Rightarrow\; R \cap \mathcal{A} \neq \varnothing.$$

In other words, the set $\mathcal{A}$ intersects all heavy ranges*, those that contain more than an ε fraction of the total points in X.*

The popular ε-net theorem is as follows [12, 13]:

Theorem 5 (ε-net Theorem) *For a range space defined like above, with VC-dimension δ, a random independent sample of size s will give an ε-net with probability more than $1 - \varphi$, where s is:*

$$s = \max\left(\frac{1}{\varepsilon}\log\frac{1}{\varphi} + \frac{\delta}{\varepsilon}\log\frac{\delta}{\varepsilon}\right) \tag{3}$$

There are two popular algorithms for building an ε-net of size $O(\frac{\delta}{\varepsilon}\log\frac{\delta}{\varepsilon})$ for a range space $(X, \mathcal{R})$ of VC-dimension δ. A randomized **sampling-based** and a deterministic **discrepancy-based** algorithm.

Sampling-based Construction. A randomized Monte-Carlo algorithm constructs an ε-net by randomly drawing a set of samples from X, based on the value s in Theorem 5.

Corollary 4 *For a constant failure probability $\varphi \leq \frac{1}{2}$, the sampling-based algorithm constructs an ε-net of size $O(\frac{\delta}{\varepsilon}\log\frac{\delta}{\varepsilon})$ in time $O(\frac{\delta}{\varepsilon}\log\frac{\delta}{\varepsilon})$.*

Indeed, a single run of the Monte-Carlo algorithm can fail to generate an ε-net. Therefore, we use it within a Las-Vegas randomized algorithm (Algorithm 2) that repeats drawing samples until an ε-net is found.

Algorithm 2 Building ε-net (Sampling-based Algorithm)

Require: The range space $(X, \mathcal{R})$, value of ε, failure probability φ, and the exponential decay m.
Ensure: The ε-net $\mathcal{A}$.

```
function BuildEpsNetSampling(X, ε, φ)                ▷ Sampling-based algorithm
    repeat
        s ← calculate the size (Equation 3)
        A ← s random samples with replacement from X.
    until IsEpsNet(A) = true
    Return A
```

Practical Implementation. Algorithm 2 relies on a function IsEpsNet that, given a set $\mathcal{A}$, verifies if it is an ε-net. Such a verification would require enumerating the collection of valid ranges, i.e., in our case, all ring ranges that contain at least ε elements ($O(n^{\delta})$). Therefore, instead of verification over the entire collection, we sample a subset of ranges, uniformly at random, and verify whether $\mathcal{A}$ contains at least one element from each of the sampled ranges. The number of sample ranges is treated as a hyperparameter in our construction algorithm.

Let T_v be the time required to verify if a set $\mathcal{A} \subset X$ is an ε-net. The expected number of time the Las-Vegas algorithm requires to repeat the sampling is $\frac{1}{1-\varphi}$. Hence, fixing the failure probability to $\varphi = \frac{1}{2}$, the expected time complexity of the algorithm is $O\left(\frac{\delta}{\varepsilon}\log\frac{\delta}{\varepsilon} + T_v\right)$.

Algorithm 3 Building ε-net (Discrepancy-based Algorithm)

Require: The range space $(X, \mathcal{R})$, value of ε, failure probability φ, and the exponential decay m.
Ensure: The ε-net $\mathcal{A}$.

function BuildEpsNetDiscV1($X, \varepsilon, \mathcal{R}$) ▷ Build ε-net by providing ε
  $k \leftarrow$ number of iterations (Equation 4)
  $\mathcal{A} \leftarrow X$
  for $1 \leq i \leq k$ **do**
    $\mathcal{A} \leftarrow$ *Halving*$(\mathcal{A}, \mathcal{R})$ ▷ The halving step, with arbitrary matching.
  Return $\mathcal{A}$
function BuildEpsNetDiscV2($X, m, \mathcal{R}$) ▷ Build ε-net by providing m
  $k \leftarrow m$ ▷ Only continue m times to reduce the size by 2^m
  $\mathcal{A} \leftarrow X$
  for $1 \leq i \leq k$ **do**
    $\mathcal{A} \leftarrow$ *Halving*$(\mathcal{A}, \mathcal{R})$ ▷ The halving step, with arbitrary matching.
  Return $\mathcal{A}$

Discrepancy-based Construction. In the following, we provide a high-level overview of the Discrepancy-based algorithm, and refer the reader to [11, 4] for more details. The algorithm (Algorithm 3) works by iteratively *halving* the point-set X, until reaching the desired size of $c_0\frac{\delta}{\varepsilon}\log\frac{\delta}{\varepsilon}$, where c_0 is a large enough constant.

In order to do so, it first constructs an arbitrary matching Π of the points. A matching Π is a set of pairs (x, y) where $x, y \in X$, it also partitions X into a set of $\frac{|X|}{2}$ disjoint pairs. Given this matching, this algorithm randomly picks one of the points in each pair, removing the other point of the pair, resulting in a subset of remaining points $X_1 \subset X$ where $|X_1| = \frac{|X|}{2}$.

Continuing this process k times for the following value of k

$$2^k = \frac{|X|}{c_0\frac{\delta}{\varepsilon}\log\frac{\delta}{\varepsilon}} \tag{4}$$

results in the set $|X_k|$ which is an ε-net for X. It is easy to make this process deterministic by following the conditional expectation method at each halving step [4]. By following a Sketch-and-Merge algorithm, one can achieve a near-linear running time for this construction [12]:

Corollary 5 *The discrepancy-based algorithm finds an ε-net of size $O(\frac{\delta}{\varepsilon}\log\frac{\delta}{\varepsilon})$ in time $O(\delta^{3\delta}\cdot(\frac{1}{\varepsilon}\log\frac{\delta}{\varepsilon})^{\delta}\cdot n)$.*

Comparison. The randomized algorithm is straightforward to implement, requiring only random sampling from each layer $\mathcal{L}_i$ to construct the subsequent layer $\mathcal{L}_{i+1}$. However, it is inherently randomized and provides running-time guarantees in expectation. Furthermore, its time complexity depends on the time to verify if the selected set is indeed an ε-net.

In contrast, the deterministic discrepancy-based algorithm deterministically constructs an ε-net by progressively halving each layer $\mathcal{L}_i$. With the hyperparameter m, as the exponential decay of the HENN graph, this process involves only halving $\mathcal{L}_i$ up to m times to identify the next layer $\mathcal{L}_{i+1}$

(See BuildEpsNetDiscV2 in Algorithm 3). Nevertheless, the running time of the discrepancy-based algorithm depends on the dimensionality of the input points. A pseudo-code of these two algorithms is provided in Algorithm 3.

B.2 Navigable Graphs

Graph-based algorithms for the ANN problem typically begin by constructing a graph on the given dataset X. A key property of these graphs is *navigability*, which ensures that the Greedy Search algorithm (Algorithm 4) can be effectively applied [30]. Specifically, navigability means that by following a sequence of locally greedy steps, the algorithm can successfully reach an approximate nearest neighbor of the query point q.

According to this definition, a *complete graph* over the point set is trivially navigable. The most optimized navigable graph can, in principle, be obtained by constructing the dual of the Voronoi diagram of the points, known as the *Delaunay triangulation* [6]. However, constructing this graph is computationally challenging, particularly in high dimensions, due to the curse of dimensionality.

An efficient approximation of the Delaunay triangulation can be achieved through a simple randomized algorithm that incrementally inserts points and connects each new point to its nearest neighbors in the existing graph structure. This approach forms the basis of the *Navigable Small World* (NSW) graph [30]. The HNSW algorithm adopts a similar strategy within each layer, while introducing additional heuristics to improve practical performance, such as adding random exploration edges between points. These heuristics can also be incorporated into the HENN structure as well, treating the navigable graph as a black-box [29].

A k-NN graph can also be used; however, it is well known that for small values of k, such graphs tend to exhibit numerous local minima [27].

As previously discussed, any black-box navigable graph constructed over the points within a single layer can be easily integrated into the HENN framework. We refer the reader to Appendix G for a comparative analysis of different navigation graph choices integrated within HENN.

B.2.1 Navigation Graph Construction via Dimensionality Reduction

We introduce a heuristic that leverages dimensionality reduction to construct a navigation graph for each individual layer of HENN. The procedure begins by reducing the dimensionality of the data points in a given layer to a low-dimensional space (typically 2D or 3D). Subsequently, we compute the exact Delaunay triangulation (DT) over the reduced representation and construct this graph on the point set. This approach is computationally efficient, as DT construction in low dimensions is fast and does not suffer from the challenges associated with high-dimensional geometric computations.

For dimensionality reduction, we employ techniques that aim to preserve pairwise distances, such as t-SNE [38]. Additional techniques and details regarding this heuristic are discussed in Appendix G.

C Dynamic Setting

In this section, we present a procedure for maintaining the HENN structure under dynamic updates to the point set X. The supported operations include `Insert(x)`, which adds a new point x, and `Delete(x)`, which removes an existing point $x \in X$.

Based on the ε-net construction described in Section B.1, a random sample of an appropriate size forms an ε-net of X with high probability. We denote this required sample size by f_ε, given by:

$$f_\varepsilon = O\left(\frac{d}{\varepsilon}\log\frac{d}{\varepsilon}\right)$$

Each layer of HENN, denoted by $\mathcal{L}_i$, is constructed as a random sample of size $f_{\varepsilon(|\mathcal{L}_{i-1}|)}$ from the previous layer (see Equation 1). Consequently, the problem reduces to dynamically maintaining a random sample S of size f from the point set X.[10]

[10] S is a random sample with replacement, with each element having a probability $\frac{f_\varepsilon}{n}$ being in S.

This problem can be addressed using Reservoir Sampling [39] and the *Backing Samples* technique [9]. The key idea is to handle `Insert(x)` operations by probabilistically adding the new element to the sample S using a non-uniform coin toss. To support deletions, a larger backing sample is maintained beyond size f, allowing for efficient resampling of S once the size drops below a threshold. This approach yields a constant amortized update time.

According to this, we can maintain the HENN structure dynamically:

`Insert(x):` To insert a new point, dynamic updates are performed starting from layer $\mathcal{L}_1$ and proceeding upward through the hierarchy, stopping at the highest layer where the new point is included. This process takes $O(\log n)$ time, matching the insertion time complexity of HNSW.

`Delete(x):` Deletion begins at layer $\mathcal{L}_1$, where the point x is removed if present. If the size of a layer falls below a critical threshold (as discussed in [9]), the layer must be resampled. Following resampling, the HENN structure is rebuilt from that layer up to the root, which incurs a cost of $O(n \log n)$ in the worst case. However, since such rebuilding occurs infrequently, only when the layer size drops significantly (e.g., $f < c_0 n$ for a constant c_0), the amortized cost remains $O(\log n)$.

D Parallelization

In this section, we present a parallelized approach to constructing the HENN index during preprocessing. While the original HENN construction, shown in Algorithm 1, runs sequentially by building layers $\mathcal{L}_1$ through $\mathcal{L}_m$ from the base point set X, this process can be parallelized to reduce preprocessing time.

To enable parallelization, we exploit the fact that layer sampling in HENN is performed with replacement. Given p parallel CPU cores, we can independently generate samples for each layer in parallel. Specifically, each core performs independent sampling, effectively achieving a p-factor speedup for the sampling phase performed on each layer.

HNSW also uses a parallelization to enhance preprocessing [29], where each input point is processed independently. For each point, a random level is assigned, and the point is inserted into the corresponding layers. This results in a total construction time of $O(n \log n)$, which can be reduced to $O(\frac{n \log n}{p})$ under parallel execution with p cores.

For HENN, the construction begins by sampling a subset of size $\frac{n}{2^m}$ for the first layer $\mathcal{L}_1$, and recursively building higher layers. Each layer $\mathcal{L}_i$ requires constructing a navigation graph, the complexity of which depends on the chosen method. For instance, NSW-based graph construction requires $O(|\mathcal{L}_i| \log |\mathcal{L}_i|)$ time on the ith layer. Excluding graph construction, the sampling phase alone can be executed in $O(\frac{n \log n}{p})$ time using p cores.

E Proofs

E.1 Proof of Lemma 2

Proof *Fix a layer i, let $s = |\mathcal{L}_{i-1}|$. The size of an ε-net according to the described procedure is $O(\frac{d}{\varepsilon} \log \frac{d}{\varepsilon})$. As a result, for the layer $\mathcal{L}_i$ and a large enough constant $c_1 < c_0$, we have:*

$$
\begin{aligned}
|\mathcal{L}_i| &\leq c_1 \frac{d}{\varepsilon(s)} \log \frac{d}{\varepsilon(s)} \\
&= c_1 \frac{s}{c_0 2^m \log s} \log \frac{s}{c_0 2^m \log s} \\
&\leq c_1 \frac{s}{c_0 2^m \log s} \log \frac{s}{2^m} \\
&= c_1 \frac{s}{c_0 2^m} \leq \frac{s}{2^m}
\end{aligned}
$$

□

E.2 Proof of Lemma 3

Proof *Since $\mathcal{L}$ is an ε-net of X, we know that each (ring) range R of size more than $\varepsilon \cdot n$, intersects with $\mathcal{L}$. In other words:*

$$|R| \geq \varepsilon n \rightarrow \mathcal{L} \cap R \neq \varnothing$$

This comes from the definition of an ε-net.

Based on the definition of Recall Bound, we know that there are at most ρ_δ more points, denoted as $P_{\mathcal{L}} = \{p_1, p_2, \cdots, p_{\rho_\delta}\}$, in $\mathcal{L}$ that are closer to q than $\overline{p}$ (with probability more than δ).

Assume that the points in $P_{\mathcal{L}}$ are sorted based on their distance to q, with p_1 being the closest. For each $p_j \in P_{\mathcal{L}}$ define a unique range R_j, which is a ring centered at q, covering all the distances between $(dist(q, p_{j-1}), dist(q, p_j))$, exclusively (see Figure 10). In addition, define one more ring, $R_{\rho_\delta+1}$ for $(dist(q, p_{\rho_\delta}), dist(q, \overline{p}))$.

All these $\rho_\delta + 1$ ranges are disjoint, and they do not contain any point in $\mathcal{L}$. Since $\mathcal{L}$ is an ε-net for this range space, this means that all the ranges R_j have at most $\varepsilon \cdot n$ points from X (the lower level). As a result, the union of all these ranges contains at most $\varepsilon \cdot n \cdot (\rho_\delta + 1)$ points.

By considering the $P_{\mathcal{L}}$ itself, this proves the fact that there are at most $\varepsilon \cdot n \cdot (\rho_\delta + 2)$ points in X that are closer to q than $\overline{p}$. □

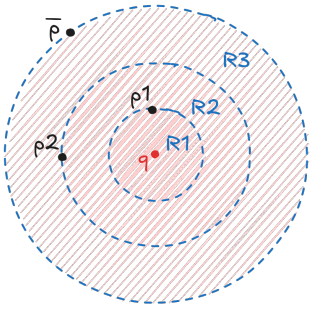


Figure 10: Visualization of Lemma 3. In this example, $\rho_\delta = 2$ and the black points are inside the ε-net.

E.3 Proof of Theorem 4

Proof *We know that the bottom-most layer $\mathcal{L}_0$ is the point set itself, and the upper layer $\mathcal{L}_1$ is an ε-net of $\mathcal{L}_0$ for $\varepsilon = c_0 d \frac{\log n}{n} 2^m$ (see Equation 1).*

The Query algorithm starts from the root and moves down to layer $\mathcal{L}_1$. Then, it finds a point $\overline{p}$ in $\mathcal{L}_1$ using the Greedy algorithm. Finally, it applies the greedy algorithm in layer $\mathcal{L}_0$, starting from the point $\overline{p}$ and returns the result.

Let $T(n)$ denote the running time of this algorithm for a point set of size n. The first step, starting from the root down to layer $\mathcal{L}_1$, is recursively equivalent to a search on a HENN graph, built on top of the points in $\mathcal{L}_1$, with the same hyperparameters. As a result, using induction, the first step takes

$$T(|\mathcal{L}_1|) = T(\frac{n}{2^m})$$

After finding $\overline{p}$ in layer $\mathcal{L}_1$, based on Lemma 3, there are at most $\varepsilon \cdot n \cdot (\rho_\delta + 2)$ points in X closer than $\overline{p}$ to q. We know that based on GreedySearch*, the second step, at each iteration, gets at least one step closer to q. Therefore, the total number of visited hops in the second step is at most $\varepsilon \cdot n \cdot (\rho_\delta + 2)$.*

Let d^* denote the degree of each node in the navigation graph $\mathcal{G}_X$. This results in the second step taking $O(\varepsilon \cdot n \cdot \rho_\delta \cdot d^*)$ time to complete. Now, we can write the following recursion on the running time $T(n)$:

$$T(n) \leq T(\frac{n}{2^m}) + \varepsilon \cdot n \cdot (\rho_\delta + 2) \cdot d^* = T(\frac{n}{2^m}) + O(d \cdot \log n \cdot 2^m \cdot \rho_\delta \cdot d^*)$$

The value 2^m is always kept as a constant (less than 64 in HNSW). Hence, using the Master Theorem, the running time would be:

$$T(n) = O(d \cdot d^* \cdot \rho_\delta \cdot \log^2 n)$$

□

F Pseudo-codes

Algorithm 4 Greedy Search Algorithm

Require: The HENN graph $\mathcal{H}$ and the query point q.
Ensure: The approximate nearest neighbor of q in X.

function QUERY($\mathcal{H}, q$)
  $v \leftarrow$ *random point from root*($\mathcal{L}_0$)
  for each layer $i = L, L-1, \cdots, 0$ **do**
    $v \leftarrow$ *GreedySearch*($\mathcal{L}_i, v, q$)
  Return v
function GREEDYSEARCH($\mathcal{G}, v, q$) ▷ $\mathcal{G}$ is the navigable graph, starting node v, and query q
  $next \leftarrow v$
  repeat
    $curr \leftarrow next$
    $\mathcal{N} \leftarrow$ neighbors of $curr$ in $\mathcal{G}$.
    $next \leftarrow \arg\min_{u \in \mathcal{N}} dist(u, q)$ ▷ Closest neighbor to q
  until $dist(next, q) \geq dist(curr, q)$ ▷ Until getting stuck in local minima
  Return $curr$

G More on Experiments

G.1 Experimental Setup Configuration

All experiments were conducted on an Ubuntu 24 server equipped with 64 CPU cores and 128 GB of RAM. Wherever applicable, we enabled hnswlib's parallel indexing capabilities for both the baseline HNSW and our proposed HENN implementations. To ensure consistent and interpretable timing results, all query operations were executed sequentially. The experiments on real-world benchmark datasets are mostly performed in Python,[11] while all other experiments were conducted in C++.

G.2 Implementation Details

The parameter m, which governs the exponential decay in layer size, is set to match the baseline configurations to ensure a fair comparison with an equal number of layers. To compute an ε-net, we draw random samples with replacement and verify whether the sampled subset satisfies the ε-net property. Otherwise, we repeat.

A naive brute-force verification would require checking coverage over all possible ranges (e.g., all rings in the specified range space), which is computationally expensive. To make this process tractable in practice, we introduce a hyperparameter r, during the indexing phase, that denotes the number of ranges to check, also referred to as the *range size*. During sampling, we randomly select r ranges from the space and ensure that the sampled points hit all of them. Additional implementation details for this phase are provided in the supplemental material (also see background on building ε-net in Section B.1 in the Appendix).

G.3 Methods and Datasets

In this section, we compare the HENN structure against the HNSW baseline. As discussed earlier, the primary distinction between the two lies in how hierarchical layers are constructed. HNSW selects layer points randomly, while HENN ensures that each layer forms an ε-net of the previous one. In

[11] We used the Python binding of the C++ implementations, similar to hnswlib.

both methods, a navigable graph is built over each layer using an incremental greedy algorithm: when a new point is added, it is connected to the approximate nearest neighbors among the existing points in that layer, forming an NSW (navigable small world) graph. Other navigable graph constructions and baselines are discussed in the supplemental material.

To evaluate performance, we use both synthetic and real-world datasets. The synthetic data is generated by varying the number of dimensions and data points, with each coordinate sampled independently from an exponential distribution parameterized by λ. Larger λ values produce more skewed distributions, making the dataset increasingly challenging and suitable for testing worst-case behavior. For real-world benchmarks, we use the SIFT [19], GloVe [35], and MNIST [24] datasets with Euclidean ℓ_2-norm. The SIFT dataset consists of 1 million 128-dimensional visual descriptors extracted from image features, accompanied by 10,000 query vectors and their ground-truth nearest neighbors. The GloVe dataset contains 50,000 pre-trained 100-dimensional word embeddings derived from global word co-occurrence statistics across large text corpora. MNIST also contains 70,000 images (images) flattened into 784-dimensional vectors.. These are popular ANN benchmarks used in the literature.[12] Additional experiments on more synthetic and real datasets are included in the supplemental material.

G.4 More Experiments

In this section, we present experiments on HENN integrated with various navigation graphs, as described in Appendix B.2. Additionally, we explore the applicability of HENN under a commonly used distance metric in approximate nearest neighbor (ANN) problems: cosine similarity (also referred to as the angular distance metric). We evaluate the performance of HENN on two real-world datasets: FashionMNIST and GloVe, with the angular metric.

Navigation Graphs. As detailed in Appendix B.2, different navigation graph constructions can be integrated into each individual layer of the HENN structure. In this section, we empirically evaluate the performance of the following navigation graph variants:

- **K-Nearest Neighbor Graph (**knn**):** Each point is connected to its exact k nearest neighbors based on the original distance metric.
- **Navigable Small World Graph (**NSW**):** This graph structure, used in HNSW, provides a navigable approximation of both the KNN and Delaunay Triangulation (DT) graphs.
- **Dimensionality Reduction-Based Graph (**dim_red**):** A dimensionality reduction technique, such as t-SNE [38], PCA, the Johnson-Lindenstrauss (JL) projection [21], or UMAP [14], is first applied to the data points, reducing them to a low-dimensional space (typically 2D or 3D). The exact DT graph is then constructed on the reduced representation.

Recall Bounds (ρ_δ). Figure 11 presents a comparison of recall bounds across different navigation graph constructions. This metric serves as a key component in the time complexity guarantees established in Theorem 4. We conduct experiments across varying dataset sizes (n) and distributions. As expected, recall bounds increase with more skewed data distributions, e.g., higher values of λ in exponential distributions, indicating a reduced probability of hitting a fixed number of nearest neighbors (k).

Notably, for both the KNN and NSW navigation graphs, the recall bound remains below 15 and exhibits minimal sensitivity to changes in dataset size. Among the dimensionality reduction-based graphs, t-SNE consistently achieves better recall than PCA, JL, and UMAP, suggesting its effectiveness in preserving neighborhood structure in low dimensions. All dimensionality reduction methods were applied with a target dimension of 3. These results highlight t-SNE's ability to solve approximate nearest neighbor search.

While KNN achieves the best recall performance, it is computationally more expensive to construct compared to other methods. NSW, despite being an approximation of KNN, shows slightly worse recall, trading off quality for efficiency.

[12] ann-benchmarks.com

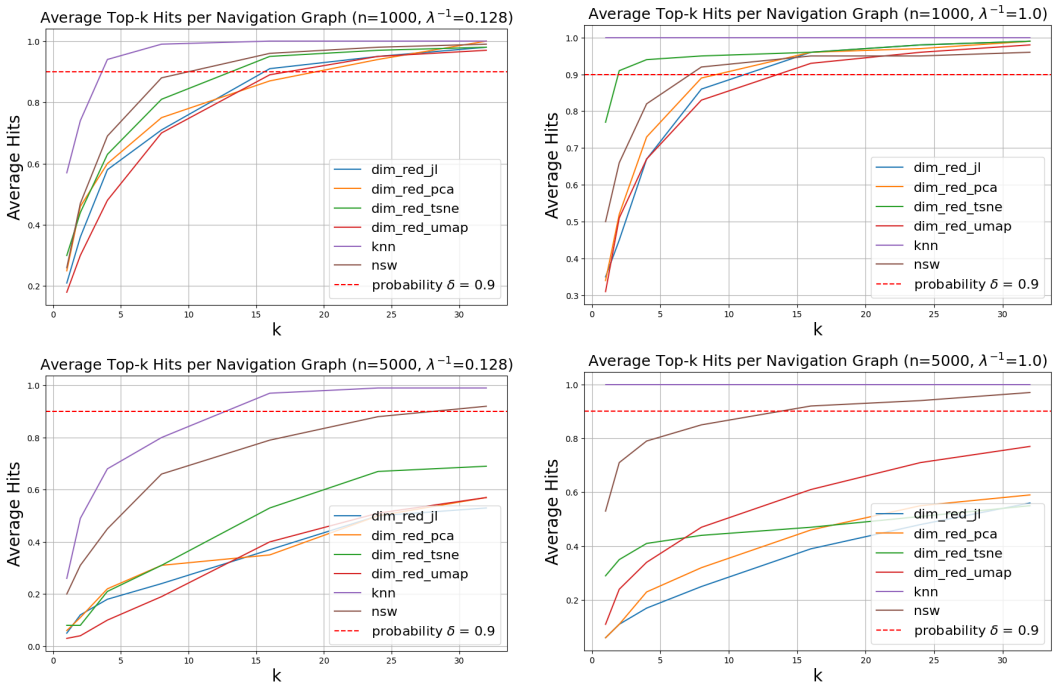


Figure 11: Comparing Recall Bound ρ_δ for different navigation graphs. *Average Hits* is the fraction of times that at least one point is retrieved within the k nearest points. The red dotted lines show the threshold for $\delta = 0.9$. The intersection of the red dotted line with the plots shows the value of $\rho_{0.9}$ for each navigation graph. The data comes from an exponential distribution with parameter λ. $\lambda = 1$ is equivalent to uniform distribution and $\frac{1}{\lambda}$ is the mean of this distribution. The dimension of the points here is 32. The values are averaged over 100 runs.

Angular Distance Metric. One widely used distance metric in approximate nearest neighbor (ANN) search is the angular (cosine) similarity metric. HENN is compatible with this metric in addition to standard ℓ_p norms. For cosine similarity, all vectors are normalized to lie on the unit sphere, and the objective is to find the vector that maximizes the cosine of the angle with a given query vector. This metric is particularly common in applications involving word embeddings, such as GloVe.

To ensure that the theoretical guarantees of HENN extend to the cosine similarity setting, we must show that the associated range space has bounded VC-dimension, similar to the case of ℓ_p norms. As discussed in Section 2, ring ranges in HENN are defined as the difference between two balls. Under cosine similarity, the analogous range corresponds to the intersection of the unit sphere with two parallel hyperplanes, representing all points whose cosine similarity with a query point q lies between two thresholds l and u. This geometric region is equivalent to a spherical stripe (or band), and it is known that the range space defined by stripes of parallel hyperplanes has VC-dimension $\Theta(d)$. Therefore, the range space under cosine similarity also has a bounded VC-dimension, and the theoretical guarantees of HENN remain valid in this setting.

HENN with Navigation Graphs. Figure 12 presents the performance of HENN when integrated with three types of navigation graphs: exact KNN, NSW (a variant similar to HNSW), and dimensionality reduction via t-SNE. Among these, the KNN-based integration consistently achieves the highest Recall@k across various values of k. HENN+NSW resembles the HNSW implementation, but in this case, we are building ε-nets for each layer instead of any arbitrary random samples.

Interestingly, for real-world datasets with inherent cluster structure, such as FashionMNIST, dimensionality reduction using t-SNE outperforms NSW, likely due to its ability to preserve local neighborhoods. In contrast, under adversarial settings such as data drawn from an exponential distribution, NSW performs more robustly. These results also demonstrate that HENN is compatible with

angular distance metrics, as both GloVe and FashionMNIST are evaluated under cosine similarity in this experiment.

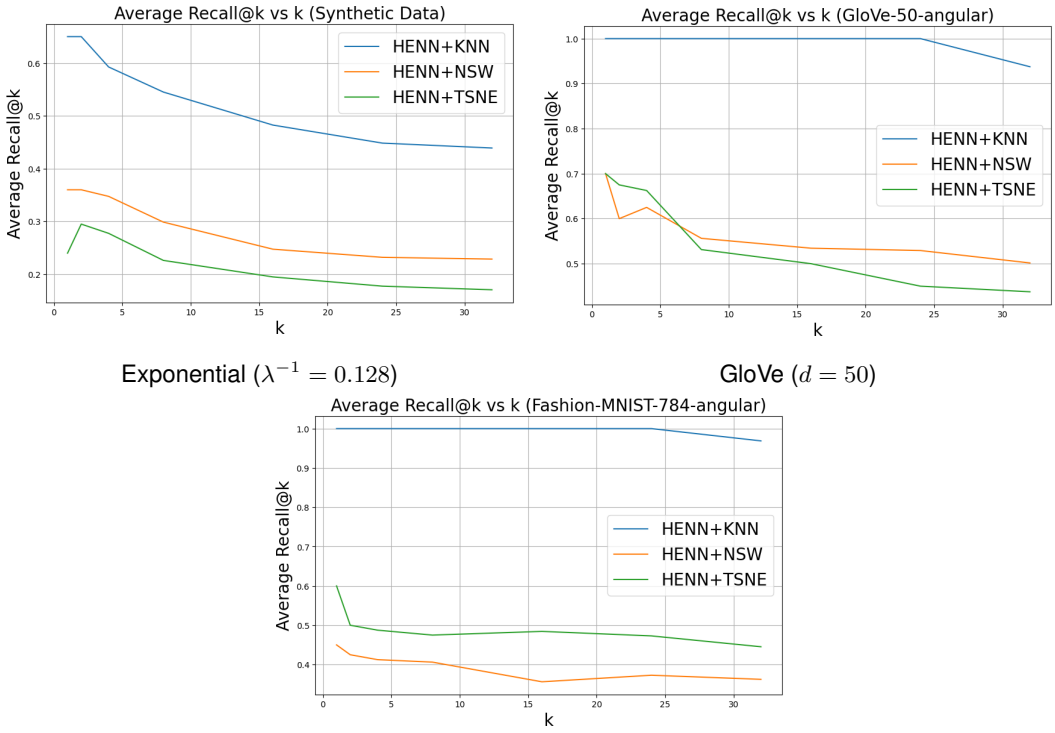


Figure 12: Comparing the Recall@k for different values of k on HENN structure integrated with different navigation graphs. The size of datasets, n, is equal to 10000 in these experiments. The distance metric on the synthetic data is the Euclidean ℓ_2-norm, while for both GloVe and Fashion-MNIST, it is angular cosine similarity.