Vista: Vector Indexing and Search for Large-scale Imbalanced Datasets

Yujian Fu*, Cheng Chen†, Yao Chen*, Weng-Fai Wong*, Bingsheng He*
*National University of Singapore, †ByteDance Inc
{yujianfu, wongwf, hebs}@comp.nus.edu.sg, yaochen@nus.edu.sg, chencheng.sg@bytedance.com

***Abstract*—With the rise of machine learning models, particularly generative models like sequence-to-vector models, there is a high demand for constructing efficient approximate nearest neighbor search (ANNS) indexes on the embedding vectors they generate. Despite the development of numerous indexes for efficient vector retrieval, the complex distributions of vectors generated by these models and their impact on ANNS tasks remain underexplored. In this work, we address the challenges faced by current advanced ANNS approaches when dealing with vectors characterized by imbalanced distributions, which negatively impact search efficiency. We identify the difficulty in indexing and searching certain vectors using previous ANNS graph indexes due to skewed distributions and propose a novel index, Vista, that improves efficiency by introducing dynamic index construction patterns based on vector distribution. Our experimental evaluation confirms Vista's efficiency advantage, demonstrating that on both public and industrial-grade real-world imbalanced datasets, Vista achieves several to tens of times performance improvement compared to advanced ANNS indexes while ensuring high search accuracy and good scalability.**

I. Introduction

k-Nearest Neighbor (k-NN) search is a fundamental task in numerous machine learning algorithms and serves as the foundation for various real-world applications such as information retrieval [36], machine learning [14], and recommendation systems [53]. With the emergence and success of large language models (LLMs) [4], [12], [35], [47], [48], nearest neighbor search has gained even more importance as a necessary component in the retrieval-augmented generation (RAG) [13], [16], [52] process. However, the curse of dimensionality [10], [26] makes vectors in high-dimensional space more indistinguishable, leading to unaffordable time costs for exact k-NN queries in modern applications. Consequently, researchers have turned to approximate solutions, trading some search accuracy for efficiency, a practice known as approximate nearest neighbor search (ANNS) [31], [32], [34], [51]. At ByteDance, achieving high accuracy and throughput in ANNS retrieval is essential for delivering precise and timely recommendations, advertisements, and risk control in large-scale, real-time applications.

Imbalanced vector distribution is a common phenomenon in ANNS datasets [11], [49], [50]. At ByteDance, we observe that embedding vectors generated by widely-used sequence-to-vector models, such as graph neural networks (GNN) [23], [25], [55]and generative pre-training models (GPT) [47], often exhibit strong imbalanced clustering properties. Examples are presented in Figure 1.

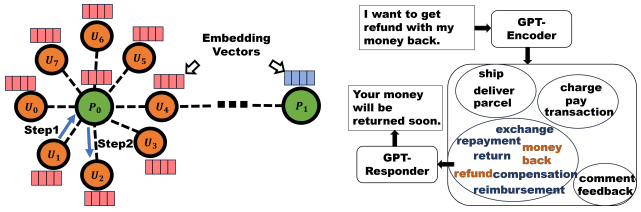
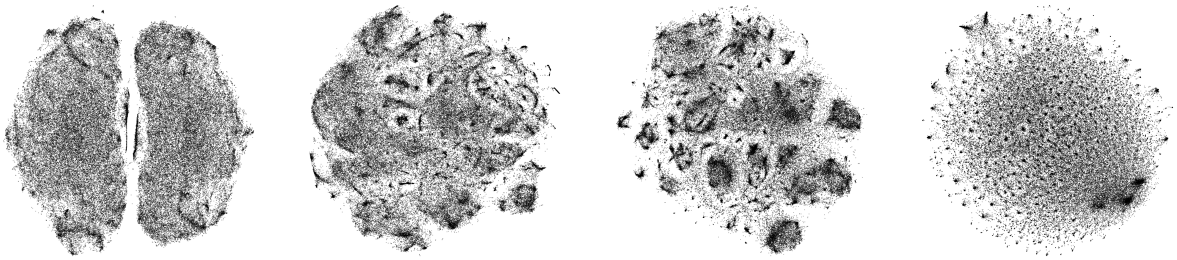


Fig. 1: Left: GNN embedding vectors bring popular products and nearby users closer together while increasing the distance to unpopular products. Right: GPT embedding vectors cluster together based on their semantic similarity.

The left figure presents the random walk GNN algorithm, which is widely used in e-commerce recommendation systems (e.g., GraphSAGE [25], DeepWalk [46], LINE [55], etc.). We connect users ($U_0 - U_7$) to purchased products (P_0, P_1). Starting from a source point (U_1), the GNN random walk generates sequences (e.g., $U_1 \rightarrow P_0 \rightarrow U_2$), forming positive pairs (e.g., $(U_1, P_0), (U_1, U_2)$) and selecting unconnected points as negatives (e.g., P_1). The algorithm clusters embeddings of positive pairs closer and pushes negatives farther away. In e-commerce, popular products (P_0) attract many users, forming dense clusters, while niche products (P_1), bought by few, have sparse, distant embeddings.

The right figure demonstrates the GPT-based language model used in online customer support. Given the input sentence 'I want to get a refund with my money back,' the GPT-Encoder tokenizes it and generates embeddings for each word. Words like 'refund' and 'money back' form a dense cluster due to frequent co-occurrence, along with related terms like 'exchange,' 'repayment,' and 'return.' In contrast, words related to Payment (e.g., 'charge,' 'pay') and Delivery (e.g., 'ship,' 'deliver') form different separate clusters, reflecting their semantic similarity. The GPT-Responder uses the dense cluster around 'refund' to generate a response like 'Your money will be returned soon.' This example highlights the clustered distribution of GPT embeddings, where similar terms group together based on their semantic context.

To understand the differences in distribution within ANNS datasets, we provide visualizations for four ANNS datasets from different sources in Figure 2, including SIFT vectors [37] for images, GoogleNET embedding vectors [54], text embedding vectors from Glove [45], and graph embedding vectors from LINE [55]. The dimensionality of the vectors is reduced to two using the t-distributed stochastic neighbor



(a) SIFT Dataset. SIFT descriptor from images. Published in 2010.

(b) DEEP Dataset. Vectors from GoogleNet for image classification. Published in 2016.

(c) GloVe Dataset. Vectors from vector representation model for text. Published in 2013.

(d) ByteECom1 Dataset. Vectors from the GNN model for e-commerce applications in ByteDance. Generated in 2024.

Fig. 2: Vector visualization of four ANNS datasets with t-SNE for dimensionality reduction. Each figure samples 100,000 vectors. We observe that vectors in the ANNS task follow an imbalanced distribution, forming both dense and sparse areas. Recent datasets, especially embedding vectors from sequence-to-vector models such as GloVe and ByteECom1, have become more skewed and generate more local small clusters.

embedding (t-SNE) [57] method, which preserves the nearest neighbor relationships between vectors. One important observation from the figure is that embedding vectors are becoming more complex, with more small local clusters, resulting in a more imbalanced distribution among the vectors. This outcome aligns with the nature of generative models based on the skip-gram model [42], which seeks to make similar items close to each other, ultimately forming groups of closely related items in various clusters within high-dimensional feature spaces. With the success of generative models for graph and text data, handling such imbalanced datasets with numerous local clusters becomes essential for large-scale ANNS tasks.

Some studies [11], [49], [50] discuss the impact of skewness in data distribution on ANNS tasks. However, such imbalanced vector distribution is rarely addressed by advanced ANNS indexes. Recent ANNS benchmarks [8], [9], [32], [58], [60] demonstrate that graph-based ANNS indexes [19], [24], [38], [39], [59] are the top performers in terms of search quality and speed, achieving high recall and throughput through fine-grained control of search paths. They deliver search speeds magnitudes faster than other ANNS methods, such as inverted indexes [29] and hashing [56], [62]. Despite extensive efforts to optimize graph structures, the impact of vector distribution on ANNS—particularly in graph index construction and retrieval performance—remains underexplored.

k-NN Connectivity	10	50	100	200
SIFT ($\times 10^2$)	8.17	6.82	6.39	6.16
DEEP ($\times 10^3$)	1.68	0.72	0.61	0.58
ByteECom1 ($\times 10^4$)	3.73	1.48	0.59	0.33

TABLE I: Vector search cost, represented by the number of distance calculations to find the vector on Vamana index [28], with respective to vector distribution, represented by k-NN connectivity on three datasets with 1 million vectors. We find vectors with lower k-NN connectivity are much more difficult to search. Detailed analysis is presented in Section III.

In this paper, we explore the imbalance in vector datasets and its impact on graph index structure and performance in ANNS tasks. This is the first study, to our knowledge, that bridges the skewness of dataset distribution with the search efficiency of ANNS indexes. We find that points with low in-degree in a k-NN graph, referred to as points with low **k-NN Connectivity** are much more difficult to discover in traditional ANNS indexes and pose a significant obstacle to achieving high recall on large-scale datasets, as shown in Table I. From the table, vectors with k-NN connectivity of 10 require significantly more distance calculations, sometimes over 10 times the search cost of other vectors.

However, advanced graph indexes [28], [40], [43], [63] ignore differences in vector distributions during index construction and use a fixed configuration for all vectors, making them inefficient at searching for these points with low k-NN connectivity. To overcome this challenge, we propose a dynamic graph index named Vista to replace the rigid patterns and configurations of previous solutions. Our dynamic pattern generates edge candidates and selects graph edges based on vector distribution. Vista compensates for nodes with low k-NN connectivity, ensuring sufficient routing edges to identify them efficiently. It mitigates the cost of retrieving ground truth vectors with poor k-NN connectivity, significantly reducing the time cost in ANNS retrieval, especially for high-recall queries. Our main contributions can be summarized as follows:

1. Identifying the challenge: We find that for imbalanced datasets, the main bottleneck for ANNS is in the retrieval of vectors with low k-NN connectivity. This is particularly important for embedding vectors from emerging sequence-to-vector models such as LLMs and GNNs, and we provide evidence for this phenomena in Section III.

2. Dynamic index design: We design a dynamic ANNS index for constructing graph edges based on vector distribution to ensure good connectivity of all vectors across the dataset, thereby reducing search costs in ANNS tasks, especially for high-recall responses.

3. Comprehensive evaluation: We implement Vista efficiently with good parallelism for index construction and retrieval. Our implementation is evaluated on public and real-world industry datasets, demonstrating its success in efficient ANNS queries and scalability in highly parallel environments.

II. Background and Related Work

A. Problem Definition

In this paper, we focus on the k approximate nearest neighbor search task for high-dimensional vectors, aiming to discover the k nearest neighbors for a given query. Since retrieving the exact nearest neighbors is often inefficient, our goal is for the ANNS index to return as many correct neighbors as possible efficiently.

B. Previous Studies

The skewness in high-dimensional vectors for ANNS tasks is observed in previous studies. M. Radovanović et al. [49], [50] discuss the frequency with which a point appears among the k nearest neighbors of other points in a dataset, confirming the common existence of points with very high k-occurrences that effectively represent 'popular' nearest neighbors across various datasets. They also point out that such 'popular' points can frequently be included in the result list during the information retrieval process. Brankica Bratić et al. [11] examine the vector distribution for the NN-descent algorithm [19] and demonstrate that points with high in-degree in a k-NN graph result in better search quality in the NN-descent index. Although the imbalance distribution of vectors has been discussed and applied to various machine learning tasks [20], advanced ANNS graph indexes are not optimized for this problem. According to the recent evaluation results on ANNS graph indexes, the following are among the most efficient solutions for ANNS in large-scale scenarios:

NSG [22] and NSSG [21] propose monotonic search networks to ensure efficient retrieval while reducing index size. HNSW [39], [40] incorporates the small-world graph [30] with the relative neighborhood graph [27] to construct a sparse hierarchical index for retrieval. HVS [38] proposes a hierarchical quantization structure for efficiently locating query neighborhoods to improve efficiency. Vamana [28] introduces a distance factor for constructing long edges on a randomly initialized graph. HCNNG [43] employs random partitions on feature space and constructs a minimum spanning tree in each partition. LSHAPG [63] combines hashing functions with a graph index to improve index construction efficiency and generate high-quality entry points for retrieval. τ-MNG [44] optimizes the edge pruning strategy to lower search time complexity when the query and its nearest neighbors are close enough. ParlayANN [17], [41] introduces an optimized parallelism strategy for the efficient implementation of various types of ANNS indexes in high-parallelism scenarios.

III. Challenge and Motivation

With emerging vector datasets becoming more imbalanced, understanding their impact on the structure and performance of graph-based ANNS indexes is crucial. This section discusses and examines how dataset distribution affects these indexes, with notations summarized in Table II.

Notation	Definition
m	Upper bound of neighbor candidates
n	Upper bound of out-edges
$\|\mathcal{X}\|$	The number of elements in set $\mathcal{X}$
$\mathcal{N}(v_i, k)$	Set of k nearest neighbors of v_i
$\mathcal{M}(v_i, k)$	Set of vectors for which v_i is among the k nearest neighbors
$\mathcal{I}(v_i)$	Set of in-edges for v_i in proximity graph
$\mathcal{S}(v_i, \mathcal{X})$	Result of edge selection from v_i to vectors in the set $\mathcal{X}$
$\mathcal{P}(v_i, k, \mathcal{X})$	Pruning result with a set of vectors selected from $\mathcal{X}$, where v_i is among the k shortest outgoing edges from $v_j \in \mathcal{X}$
$\mathcal{R}(v_i, v_j, \mathcal{X})$	The rank of the distance between v_i and v_j among the distances from vectors in $\mathcal{X}$ to v_i, ordered from smallest to largest

TABLE II: Summary of notations.

A. Analysis on Graph Index and Imbalanced Distribution

To facilitate this analysis, Table III summarizes the construction process of advanced graph-based indexes, emphasizing the impact of dataset distribution. A recent benchmark [60] identifies five key components: initialization, edge candidate acquisition, edge selection, seed preprocessing, and connectivity assurance. We focus on the two most critical components influenced by vector distribution: edge candidate acquisition and edge selection.

For all graph indexes, edge candidate acquisition involves generating a subset of vectors close to each database vector v as potential linking candidates. Strategies vary across algorithms: many, including HNSW [40], NSG [22], and Vamana [28], use v as a query vector on the initialized index, retaining the closest vectors (HNSW, NSG) or all visited nodes (Vamana). HCNNG [43] partitions the dataset, treating vectors in the same partition as candidates. Edges are then selected using distance-based pruning rules, such as the relative neighborhood graph (RNG) in HNSW and Vamana, monotonic RNG in NSG, and minimum spanning tree (MST) in HCNNG. Some indexes, like HNSW and Vamana, also add reverse edges to ensure bi-directional connections, subject to an out-degree limit.

From the discussed edge construction process, in-edges for a vector v_i come from two sources: (1) reverse edges from its selected edge neighbors and (2) out-edges from vectors that select v_i as a neighbor based on edge selection strategies. The first type of edges is represented by $\mathcal{S}(v_i, \mathcal{N}(v_i, m))$, indicating reverse edges from neighbors of vector v_i based on a specific edge selection strategy. The second type is represented by $v_j \mid v_i \in \mathcal{S}(v_j, \mathcal{N}(v_j, m))$, denoting vectors v_j that select v_i as a neighbor, making v_i an edge candidate. The final set of in-edges is the union of these two sets, clipped to the upper bound of n out-edges. i.e.

$$\begin{aligned}\mathcal{I}(v_i) = \mathcal{P}(v_i, n, (\mathcal{S}(v_i, \mathcal{N}(v_i, m))))\\ \cup \mathcal{P}(v_i, n, \{v_j \mid v_i \in \mathcal{S}(v_j, \mathcal{N}(v_j, m))\})\end{aligned} \quad (1)$$

From Equation 1, the number of in-edges for a vector is primarily affected by how many vectors consider it as a neighbor. This depends on: (1) the pruning process for both

Graph Index	Year	Initialization	Edge Candidate Acquisition	Candidate Size Control	Edge Selection Strategy	Edge Size Control
HNSW [40]	2018	Random Seed and Incremental Insertion	Search	Fixed Bound	Relative Neighborhood Graph (RNG)	Fixed bound
Vamana [28]	2019	Random Regularized Graph	Search	All Visited Nodes	RNG and Distance Parameter	Fixed Bound
HCNNG [43]	2019	Random Partition Trees	Vector Pairs in Partitions	Fixed Bound	Minimum Spanning Tree	Fixed Bound
NSSG [21]	2021	k-NN Graph	Search	Fixed Bound	Monotonic Network and Angle Parameter	Fixed Bound
LSHAPG [63]	2023	LSH Partition	Search	Fixed Bound	k-NN Graph	Fixed Bound
Vista	2024	Random Partition Trees and Approximate k-NN Graph	**Optimized Search**	**Dynamic Bound based on Distribution**	RNG	**Dynamic Edge Exchange**

TABLE III: Index construction flow of advanced graph-based methods. Vista employs a dynamic index pattern tailored to vector distribution to address vector skewness.

edge types, denoted as $\mathcal{P}(v_i, n, \mathcal{X})$, requiring v_i to be an n-closest edge candidate for vectors within $\mathcal{X}$, and (2) the number of vectors that select v_i as a neighbor, impacting the second type of edges. In the absence of edge pruning ($\mathcal{S}(v, \mathcal{X}) = \mathcal{X}$), the index becomes a bi-directional k-NN graph. For each vector v_i, bi-directional edges are formed with $\mathcal{N}(v_i, m) \cup \mathcal{M}(v_i, m)$, leading to the in-degree of vector v_i:

$$\mathcal{I}(v_i) = \{v_j \in \mathcal{N}(v_i, m) \cup \mathcal{M}(v_i, m) \mid \mathcal{R}\left(v_j, v_i, \mathcal{N}(v_j, m) \cup \mathcal{M}(v_j, m)\right) \leq n\} \quad (2)$$

As indicated by Equation 2, this pattern results in an imbalance in vector connectivity when the vector distribution is skewed. For vector v_i, if $|\mathcal{M}(v_i, m)|$ is small or zero, few or no vectors will attempt to connect to it. Moreover, the reverse edge from $v_j \in \mathcal{N}(v_i, m)$ may be pruned in favor of shorter edges in $\mathcal{N}(v_j, m) \cup \mathcal{M}(v_j, m)$ due to the upper bound n, leading to poor connectivity for v_i. Such imbalances in vector distribution are commonly observed in high-dimensional datasets, as noted in several studies [11], [49], [50]. To guarantee at least one in-edge for each vector, the minimum upper bound on the out-degree across the entire dataset is:

$$n_{min} = \max_{v_i \in \mathcal{V}} \left(\min_{v_j \in \mathcal{N}(v_i, m) \cup \mathcal{M}(v_i, m)} \left(\mathcal{R}\left(v_j, v_i, \mathcal{N}(v_j, m) \cup \mathcal{M}(v_j, m)\right)\right)\right) \quad (3)$$

In imbalanced datasets, Equation 3 highlights a potential conflict between maintaining vector connectivity and enforcing a fixed upper bound on out-degrees. Isolated points like v_i, which are far from all neighbors v_j, require many edges to connect vectors between v_j and $\mathcal{N}(v_j, m) \cup \mathcal{M}(v_j, m)$ when their distances are smaller than that between v_i and v_j. This necessitates a large out-degree bound to maintain v_i's connectivity. Variants of bi-directional k-NN graphs with specific edge selection strategies encounter similar connectivity challenges due to fixed out-degree limits.

In high-dimensional, large-scale datasets, maintaining strict out-degree limits while ensuring connectivity is often impractical due to computational constraints, with typical bounds set to tens. To guarantee a search path to all vectors, some indexes, like NSG [22], add edges to vectors lacking in-edges. However, this can exceed the predefined out-degree limit, leading to unavoidable hubs—vectors with a disproportionately large number of out-edges.

B. Evaluation on the Impact of Imbalanced Distribution

In this work, we refer to the **in-degree of a vector in a k-NN graph**, i.e. $|\mathcal{M}(v_i, k)|$, as its '**k-NN connectivity**' to this particular in-degree from that of other graphs present in the workflow. To assess the impact of vector imbalance on index structure and performance, we conducted preliminary experiments on three ANNS datasets: SIFT, DEEP, and ByteECom1, each consisting of 1 million vectors, using the Vamana [28] graph index. The maximum number of edges and the size of the search list during index construction were both set to 50. As we will show, k-NN connectivity is an important proxy for performance slowdown.

Distribution of k-NN Connectivity. First, we explore the distribution of k-NN connectivity on ANNS datasets. In this experiment, we set k equal to 50. The histogram of 50-NN connectivity distribution is shown in Figure 3. From the results, we observe that the k-NN connectivity follows a long-tail imbalanced distribution. Specifically, many vectors have relatively low k-NN connectivity. For instance, in all datasets, the connectivity of about 40% points is under 30, while about 10 percent of points are well connected, with an in-degree higher than 100 in the k-NN graph. Comparing the three datasets, the ByteECom1 dataset contains more vectors with an in-degree below 50, whereas the SIFT and DEEP datasets have more vectors with k-NN connectivity over 100.

Impact of k-NN Connectivity on Index Structure. Next, we examine how such an imbalance in k-NN connectivity affects the edge structure of the graph index. We analyze the number of in-edges from its 50 nearest neighbors in the constructed Vamana graph concerning the k-NN connectivity of points. The long edges are not included as they are mainly used for navigating to different regions of the whole index, and the search process from the neighborhood of the query takes the majority of search time [33]. The examination results are shown in Figure 4 with scatters. For clear visualization, each scatter point represents the centroid of 1000 nearby points.

In all datasets, the number of in-edges from the 50 nearest neighbors increases with the k-NN connectivity. This supports our analysis of graph index construction: vectors with higher k-NN connectivity tend to receive more links from their neighbors, as described by Equation 1. Based on this observation,

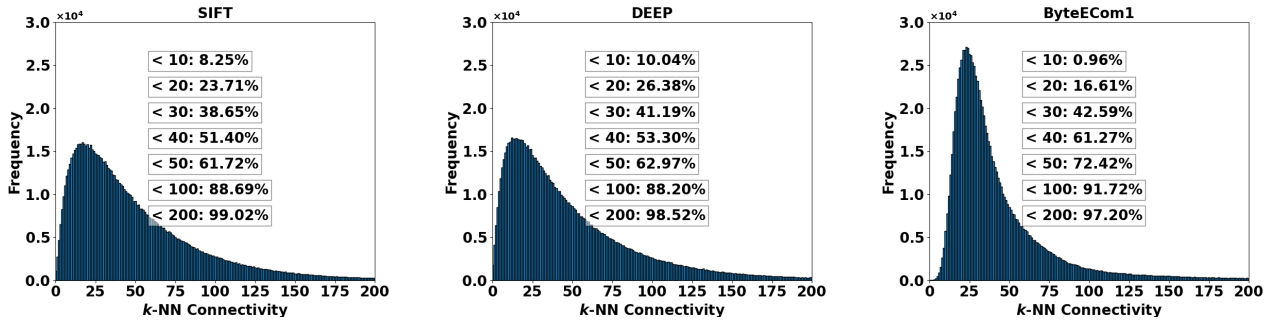


Fig. 3: k-NN connectivity distribution in a 50-NN graph on ANNS datasets. k-NN connectivity for each point indicates the number of vectors that consider it an edge candidate. The percentages of points with k-NN connectivity lower than certain bounds are plotted in the figure. We observe the distribution of k-NN connectivity is imbalanced in all datasets.

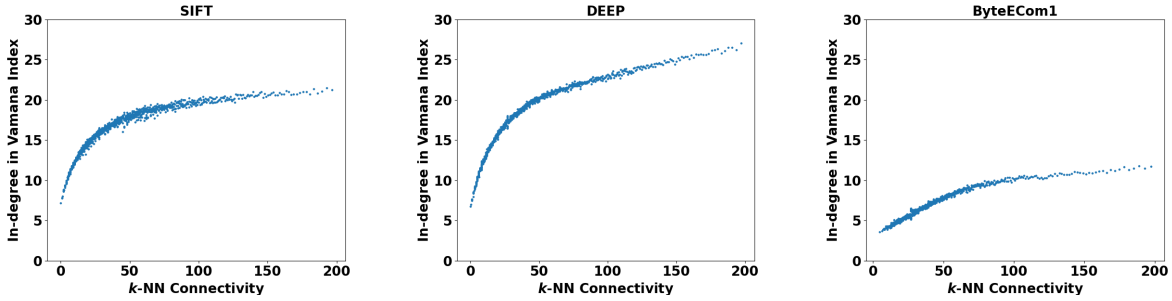


Fig. 4: In-degree from 50 nearest neighbors in Vamana graph with respect to the k-NN connectivity of points. We observe an increment in the number of in-edges from neighbors in Vamana with the k-NN connectivity of points.

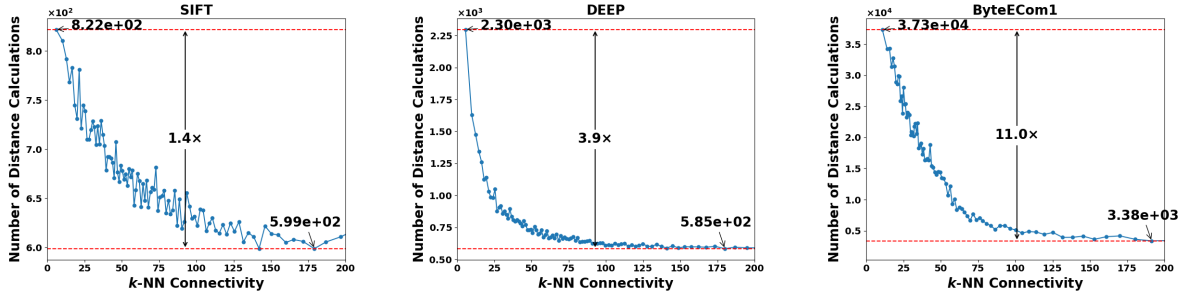


Fig. 5: Minimum number of distance calculations for searching ground truth vectors with respect to their k-NN connectivity. Vectors with higher k-NN connectivity are easier to find, while those with lower connectivity are more difficult to retrieve.

we expect that well-connected vectors will be more easily retrieved from their neighboring vectors, whereas vectors with poor connectivity will be harder to retrieve, which aligns with previous findings [11], [49].

Impact of k-NN Connectivity on Search Efficiency. We further explore the search cost of vectors by launching retrieval tasks with 10,000 query vectors and testing the minimum search cost for identifying ground truth vectors. In Figure 5, we report the minimum number of distance calculations required to find the ground truth vector as an indicator of search cost. The ratio between the highest and lowest search costs is plotted in the figure. Across the three evaluated datasets, the search cost for vectors continually decreases with k-NN connectivity due to more in-edges from their neighbors, as we analyzed. Vectors in ByteECom1 demonstrate an impressive difference, with an 11-fold variation in search cost. Therefore, we confirm that vectors with low k-NN connectivity are much more expensive to retrieve than other points.

Importantly, besides achieving high recall necessitating the search for vectors with low k-NN connectivity, which is inherently expensive as shown in previous figures, it also increases the cost of retrieving vectors with high k-NN connectivity. This is because searching for poorly connected vectors requires a large search list size, which is global for all queries. Consequently, the cost of retrieving well-connected vectors also rises. In Figure 6, we present the comparison

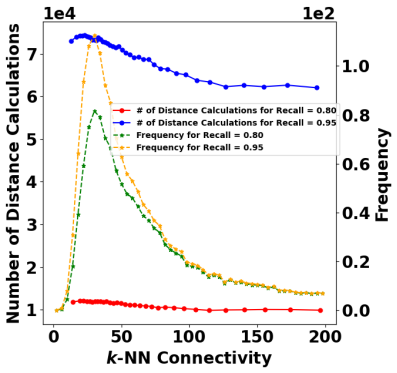


Fig. 6: Comparison of the distribution of retrieved ground truth vectors and search cost at different Recall10@10 (defined in Section V) on the ByteECom1 dataset. The result shows that achieving high recall performance requires searching for more vectors with low k-NN connectivity. This increased search effort for hard-to-retrieve vectors also raises the cost of searching vectors with high k-NN connectivity, due to the global configuration of search list size.

of ground truth distribution and search cost between different recall performances. With higher recall performance at 0.95, more vectors with low k-NN connectivity need to be searched, as indicated by the gap between green and orange lines. Here, we observe a clear increase in the frequency of retrieved vectors with low k-NN connectivity (e.g., smaller than 50). Meanwhile, the search cost, represented by the number of distance calculations, increases for all vectors. Even though we can search well-connected vectors with relatively low computation costs, as shown by the red curve, the demand for retrieving hard-to-retrieve vectors with low k-NN connectivity increases the overall search cost by over sixfold due to the larger search list size in beam search, as shown by the blue curve. Thus, searching for points with low k-NN connectivity is the primary bottleneck in efficiency.

From our discussion and evaluation in this section, we complete the analysis of the impact of vector distribution, represented by k-NN connectivity, on graph index structure (Figure 4) and search efficiency (Figure 5 and Figure 6). It is observed that vectors with a lower in-degree in the k-NN graph, i.e., lower k-NN connectivity, tend to have a smaller in-degree in the constructed index from their neighbors, making them more difficult to identify as ground truth in retrieval. Additionally, new emerging datasets, such as those generated by models on text and graph data like ByteECom1, are more skewed than others and are more likely to suffer from these challenges. This makes the problem more pronounced in modern and future applications.

IV. Index Design

Most advanced graph-based indexes use rigid patterns and configurations for all vectors during edge candidate generation and edge selection, disregarding the varying nature of vector distributions, as shown in Table III. However, as discussed in Section III, maintaining a fixed upper bound while ensuring

Algorithm 1 Vista Index Construction Algorithm

Input: Dataset $\mathcal{V}$, Max leaf size in RPT L, Number of RPTs R, Number of nearest neighbors in initialization K
Output: Constructed Graph Index $Edges$

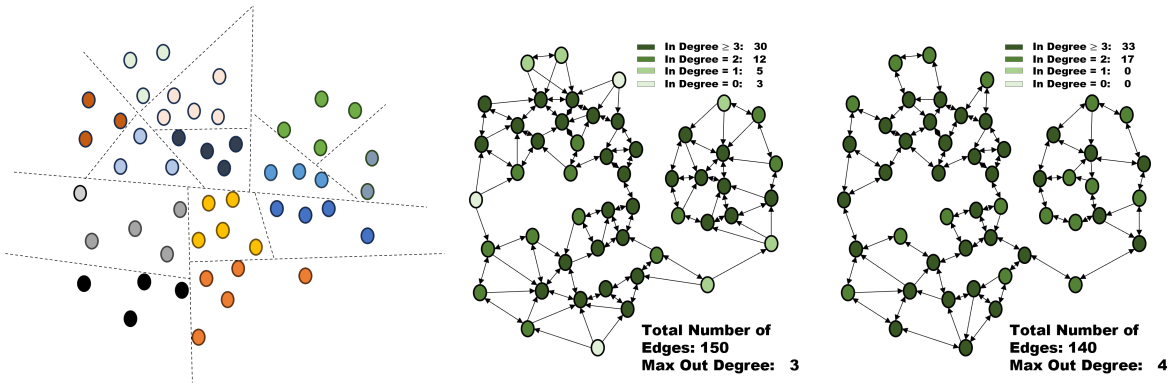
1. Initialization
$RPTs \leftarrow$ CONSTRUCT-RPT($\mathcal{V}$, L, R)
$InEdges, OutEdges \leftarrow$ INITIALIZE-NN($\mathcal{V}$, K, $RPTs$)
2. Edge Candidate Acquisition & Edge Selection
Get α with Equation 5
Get one random projection tree: $RPT \leftarrow RPTs[0]$
for v_i in $\mathcal{V}$ **do**
  Get C_i with Equation 4
  $Candidates \leftarrow$ BEAM-SEARCH(v_i, RPT, $InEdges$, $OutEdges$, C_i, K, α)
  $InEdges, OutEdges \leftarrow$ UPDATE-EDGE(v_i, $Candidates$, $InEdges$, $OutEdges$, K)
end for
for v_i in $\mathcal{V}$ **do**
  $Edges[v_i] \leftarrow InEdges[v_i] \cup OutEdges[v_i]$
end for
3. Dynamic Edge Exchange
for v_i in $\mathcal{V}$ **do**
  if $|Edges[v_i]| > K$ **then**
    $Edges \leftarrow$ EXCHANGE-EDGE(v_i, $Edges$)
  end if
end for

vector connectivity often leads to conflicts, and vectors with low k-NN connectivity are significantly harder to retrieve due to fewer in-edges from their neighbors. Therefore, optimizing the graph index construction process specifically for hard-to-retrieve vectors with a distribution-aware approach could be highly beneficial. In this section, we introduce our dynamic index design, Vista, based on the distribution of k-NN connectivity. Vista aims to construct more edges for hard-to-retrieve vectors that likely have a low in-degree in traditional edge construction patterns. The core idea can be summarized as follows: 1. Generate more neighbor candidates to build edges to points with low k-NN connectivity. 2. Prevent the deletion of necessary edges linked to vectors with low k-NN connectivity.

A. Algorithm Description

The entire index construction flow of our design is summarized in Algorithm 1. Our dynamic index construction starts with an estimation process that captures the k-NN connectivity distribution of database vectors. As we aim to create efficient indexes for large-scale datasets, such as those exceeding 100 million vectors, calculating the accurate k-NN relationship to compute connectivity for large-scale vectors is prohibitively expensive. Instead, we employ an approximation process to estimate the k-NN structure efficiently.

Initialization and Beam Search (Algorithm 2, 3, 4): We first construct multiple random projection trees (RPTs) [15] to partition the original dataset into disjoint subspaces with hyperplanes, as described in Algorithm 2, with control on the maximum size of each partition. All vectors within the same partition are potential k-NN neighbors to each other, so we calculate the distances between them and preserve the K closest neighbors for all vectors as approximate results. With a sufficient number of RPTs (e.g., 32), we discover K approximate neighbors for all vectors as an initialization of



(a) Space partition with random partition tree. (b) Initialization with 3-NN graph. (c) Vista index construction result.

Fig. 7: An example of our index construction process in 2-dimensional space with 50 vectors. Figure (a): the feature space is partitioned into disjoint subspaces using RPT, with vectors in the same partition (indicated by the same color) considered as potential neighbors. Figure (b): the generated k-NN graph with $k = 3$ for initialization, where points are color-coded based on their in-degree; darker colors indicate higher in-degree. Figure (c): the final index construction of Vista, where light-colored points with low in-degrees (e.g., ≤ 1) are eliminated and the total number of edges is reduced from 150 to 140.

Algorithm 2 CONSTRUCT-RPT($\mathcal{V}, L, R$)

Input: Dataset $\mathcal{V}$, Max leaf size in RPT L, Number of RPTs R
Output: Constructed random projection trees $RPTs$

$RPTs \leftarrow \emptyset$
for $i = 1, \cdots, R$ **do**
  $RPTs \leftarrow RPTs \cup$ RECURSIVE-PARTITION($\mathcal{V}$)
end for

function RECURSIVE-PARTITION($\mathcal{X}$)
  if $|\mathcal{X}| < L$ **then**
    return $[\mathcal{X}]$
  end if
  Randomly select two vectors v_1 and v_2 in $\mathcal{X}$
  Split $\mathcal{X}$ into two subsets $\mathcal{X}_1$ and $\mathcal{X}_2$, where vectors are closer to v_1 or v_2 respectively
  return [RECURSIVE-PARTITION($\mathcal{X}_1$), RECURSIVE-PARTITION($\mathcal{X}_2$)]
end function

Algorithm 3 INITIALIZE-NN($\mathcal{V}, K, RPTs$)

Input: Dataset $\mathcal{V}$, Number of nearest neighbors in initialization K, Random projection trees $RPTs$
Output: Out-edges of vectors $OutEdges$ and In-edges of vectors $InEdges$

for $v_i \in \mathcal{V}$ **do**
  $OutEdges[v_i] \leftarrow \emptyset$
  $InEdges[v_i] \leftarrow \emptyset$
end for
for $i = 1, \cdots, R$ **do**
  for each partition in i-th RPT **do**
    for each vector pair (v_i, v_j) in the partition **do**
      Update $OutEdges[v_i]$ with v_j, keep at most K closest vectors
      Update $OutEdges[v_j]$ with v_i, keep at most K closest vectors
    end for
  end for
end for
for $v_i \in \mathcal{V}$ **do**
  for $v_j \in OutEdges[v_i]$ **do**
    $InEdges[v_j] \leftarrow InEdges[v_j] \cup v_i$
  end for
end for

our graph index, as outlined in Algorithm 3. Additionally, we keep one random projection tree for further index construction and retrieval using the beam search algorithm, as described in Algorithm 4. The space partition result of the RPT provides a high-quality starting neighborhood for queries. For a given query vector, we first assign it to RPT space partitions and start the query process with the vector closest to the centroid of all vectors in the assigned partition. As vectors with low k-NN connectivity are hard to retrieve with out-edges, we maintain an additional structure that keeps the in-edges of all vectors.

An example of this process is presented in Figure 7 (a), where the vectors in different partitions are plotted with different colors. The distances between vectors of the same color are calculated, and we keep the K nearest neighbors for all vectors. Generating more RPTs helps improve the accuracy of the k-NN relationship. Figure 7 (b) provides an example of a 3-NN graph of our initialization result, colored by k-NN connectivity. There are many points with in-degrees less than 2, and some points have no in-edges at all, meaning they are never retrieved through the graph.

Edge Candidate Acquisition (Algorithm 4): We continue the edge candidate acquisition based on the constructed approximate k-NN graph. To generate edge candidates, we search each vector as a query to identify its close neighbors as candidates. To improve the quality of candidates, we retrieve both out-edges and in-edges in the approximate k-NN graph. Our analysis of k-NN connectivity indicates that points with low k-NN connectivity have very few or even no in-edges in the initial index, making them very difficult, or even impossible, to retrieve if we only search the out-edges, thus we need to search the in-edges. Since the in-degree of vectors is not controlled and some points have a very large in-degree, we randomly sample K in-edges for retrieval if the number of in-edges exceeds K, in order to improve construction efficiency.

Moreover, most proposed graph indexes use a fixed bound for the number of candidates, while our analysis shows that vectors with low k-NN connectivity are more challenging to

Algorithm 4 BEAM-SEARCH($q, RPT, InEdges, OutEdges, C, K, \alpha$)

Input: Query vector q, Random projection tree RPT, Out-edges of vectors $OutEdges$ and In-edges of vectors $InEdges$, Size of search list C, Maximum number of in-edges to visit K, Parameter for calculating number of result neighbors α
Output: $\mathcal{C}$ with $\alpha \cdot C$ closest neighbors to q

Assign q to space partitions in RPT, and v is the vector in the assigned partition closest to the centroid
$\mathcal{L} \leftarrow v$
$\mathcal{C} \leftarrow v$
while $\mathcal{L}$ includes unexpanded point(s) **do**
  $p \leftarrow$ closest unexpanded node in $\mathcal{L}$
  Mark p as expanded
  Update $\mathcal{L}$ and $\mathcal{C}$ with $OutEdges[p]$
  Update $\mathcal{L}$ and $\mathcal{C}$ with at most K elements sampled from $InEdges[p]$
  Keep at most C closest vectors to q in $\mathcal{L}$
  Keep at most $\alpha \cdot C$ closest vectors to q in $\mathcal{C}$
end while

retrieve with traditional indexes. Thus, we assign a larger candidate budget for vectors with low in-degree in the approximate k-NN graph to balance the difficulty in retrieval. Given a vector v_i with k-NN connectivity p_i, its size of the search list in the beam search process is calculated as:

$$C_i = K + (K - \min(K, p_i)) \tag{4}$$

As indicated by this function, we provide linear compensation for vectors with an in-degree in the approximate k-NN graph lower than K. For vectors with k-NN connectivity greater than K, their search list size is set to K. Conversely, for vectors with k-NN connectivity less than K, their search list size is adjusted to $2 \cdot K - p_i > K$. This approach enables these vectors to construct more in-edges from their neighbors in the Vista index.

To generate a sufficient number of candidates for edge selection, we preserve the closest $\alpha \cdot C_i$ vectors as edge candidates for further selection. Here, α is a parameter learned from the relative neighborhood graph (RNG) [7], [27] edge selection process in the given dataset. From the generated approximate k-NN relationship with K neighbors, we sample a small subset of the dataset (e.g., 1,000 vectors) and apply the RNG edge pruning algorithm to their K neighbors. If the average number of preserved edges per vector is P, then:

$$\alpha = K/P \tag{5}$$

By learning the distribution-specific parameter α, we ensure a sufficient number of edge candidates while maintaining efficiency by avoiding an excessive number. If the RNG process exhibits a linear relationship, we expect to retain C_i edge candidates for vector v_i after edge selection.

Edge Selection and Update (Algorithm 5): With generated candidates for all vectors, we construct edges between each vector and its candidates using the RNG principle. Candidates are sorted by their distance to the vector, and selection starts from the closest. According to the RNG rule, a candidate is chosen only if it is closer to the base vector than any of the already selected candidates. Once all candidates are evaluated, we create edges from the selected candidates to the vector. Additionally, edges are formed from the vector to its selected candidates if the candidate is among the closest K edge candidates.

Algorithm 5 UPDATE-EDGE($v, Candidates, InEdges, OutEdges, K$)

Input: Database vector v, Edge candidates $Candidates$, In-edges of vectors $InEdges$, Out-edges of vectors $OutEdges$, Upper bound of $OutEdges$ size K
Output: Updated in-edges $InEdges$ and out-edges $OutEdges$

Initialize selected candidates $\mathcal{C} \leftarrow \emptyset$
while $Candidates$ is not empty **do**
  $p \leftarrow$ closest vector in $Candidates$ to v
  Remove p from $Candidates$
  if p is closer to v than any vectors in $\mathcal{C}$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{p\}$
  end if
end while
for $v_i \in OutEdges[v]$ **do**
  Remove v in $InEdges[v_i]$
end for
for $v_i \in \mathcal{C}$ **do**
  $InEdges[v_i] \leftarrow InEdges[v_i] \cup v$
end for
$OutEdges[v] \leftarrow$ at most K closest elements to v in $\mathcal{C}$

Algorithm 6 DYNAMIC-EXCHANGE-EDGE($v, Edges$)

Input: Database vector v, Out-edges of vectors $Edges$
Output: Updated out-edges of vectors $Edges$

Initialize preserved edges $\mathcal{C} \leftarrow \emptyset$
while $Edges[v]$ is not empty **do**
  $p \leftarrow$ closest vector in $Edges[v]$ to v
  Remove p from $Edges[v]$
  $ExchangeFlag \leftarrow$ False
  for $s \in \mathcal{C}$ **do**
    if p is closer to s than v and $|Edges[s]| < |\mathcal{C}| + |Edges[v]|$ **then**
      $Edges[s] \leftarrow Edges[s] \cup p$
      $ExchangeFlag \leftarrow$ True
      break
    end if
  end for
  if $ExchangeFlag$ is False **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup p$
  end if
end while
$Edges[v] \leftarrow \mathcal{C}$

Notably, there is no fixed upper bound for controlling the maximum number of final out-degrees in our edge selection process. The upper bound K applies only to the set $OutEdges$, while we maintain an independent set $InEdges$. The final edge structure is based on the union of these two sets (line 13 in Algorithm 1). There is no fixed bound for the $InEdges$ set, and we expect vectors with low k-NN connectivity to acquire more in-edges from this set due to their larger candidate set size based on Equation 4.

Dynamic Edge Exchange (Algorithm 6): As pointed out in Equation 3, incorporating a fixed upper bound on the final out-degrees often conflicts with ensuring the connectivity of points, particularly for vectors with low k-NN connectivity. However, constructing edges without bound on the out-degree can lead to hubs, which are vectors with excessively large out-degrees within the graph index. These hubs can introduce unnecessary search costs due to expansions from them.

In Vista, we employ a dynamic edge exchange process based on vector distribution to mitigate the problem of hubness. After the edge selection process, we replace the out-edges of hub

vectors with edges between their out-neighbors to reduce their out-degrees while maintaining the RNG condition, thereby ensuring efficient routing properties. Specifically, we examine the out-edges of all vectors with out-degrees exceeding K. If an edge can be pruned by using an intermediate point with a lower out-degree, we replace the original out-edge with an edge from the intermediate point to the target. This approach helps to maintain efficiency while reducing unnecessary connectivity from hub nodes.

The main difference between our design and traditional edge pruning strategies with fixed bounds is that we do not directly delete the edges. Instead, we replace the edge with another one based on the RNG condition and out-degree distribution, a process we call edge exchange. This pattern protects the edges connected to poorly connected points, ensuring there are still sufficient edges linked to them after the exchange. We present an example of our index construction result in Figure 7 (c). Compared with the initialization result in Figure 7 (b), our index prunes the edges using the RNG rule to remove long edges, thus reducing the total number of edges and generating a sparse graph. We also ensure there are sufficient edges to points with low k-NN connectivity, reducing their retrieval cost by excluding points with low in-degrees from neighbors, e.g., $<= 1$. Although our index does not control the maximum out-degree with a fixed bound, resulting in the maximum out-degree increasing from 3 to 4, our edge exchange pattern introduced in Algorithm 6 helps mitigate this problem.

B. Complexity Analysis

Let N be the total number of vectors in the dataset. We analyze the complexity of the Vista index as follows:

Search Complexity: The time complexity for the retrieval task is the same as other variants of RNG, such as HNSW and NSG, i.e., $O(\log(N))$.

Construction Complexity: The construction complexity is the sum of the complexities of all stages. The initialization process, including partitioning the dataset and generating approximate k nearest neighbors, has a complexity of $O(n \cdot (N \cdot \log(\frac{N}{m}) + N \cdot m))$. The edge candidate acquisition requires searching the entire dataset for each vector, resulting in a complexity of $O(N \cdot \log(N))$. Selecting and exchanging the edges with the RNG rule takes $O(N \cdot (C + \alpha \cdot k)^2)$. Overall, the complexity of building Vista is the sum of all components, i.e., $O(N \cdot (n \cdot \log(\frac{N}{m}) + n \cdot m + \log(N) + (C + \alpha \cdot k)^2)) \approx O(N \cdot \log(N))$.

V. Experiments

We present a comprehensive evaluation of Vista, comparing it with leading graph-based indexes for large-scale ANNS. Vista demonstrates efficient, high-quality retrieval with affordable construction costs across public, synthetic, and ByteDance industry datasets. We assess its scalability across varying dataset sizes, explore parameter settings for cost-efficiency trade-offs, and confirm its optimization for retrieving challenging vectors. These results highlight Vista's performance and competitiveness in ANNS, with more details in our technical report.

A. Experimental Setting

Datasets. In our evaluation, we include datasets from five sources, varying in scale from 1 million to 1 billion vectors, as shown in Table IV. The DEEP dataset consists of embedding vectors from GoogleNet [54] for image classification. We conduct experiments on varying scales, ranging from 1 million to 1 billion vectors from the DEEP embeddings. The smaller-scale subsets are randomly sampled from the public 1 billion vector dataset. ByteECom1 and ByteECom2 are datasets used in ByteDance's real-world e-commerce scenarios. These embedding vectors are generated from an e-commerce graph with billions of nodes and approximately 50 billion edges using the GNN algorithm LINE [55]. For the stress test on vector skewness, we generate million-scale synthetic datasets with varying distributions. The Synthetic Uniform dataset contains 1 million vectors with dimensions uniformly distributed as $U(-1, 1)$. For Gaussian-distributed vectors, we generate 1 million vectors with 1 to 100 clusters, where 'Synthetic A' represents A clusters. Each cluster's center is defined by its binary ID (e.g., cluster 1: [0, ..., 1], cluster 2: [0, ..., 1, 0]), with equal-sized clusters and data points following a Gaussian distribution (variance = 1.0) in each dimension. This models real datasets with multiple local clusters.

All datasets in the evaluation employ 10,000 queries for retrieval, which are not included in the database vectors. We include the local intrinsic dimension (LID) [5] of the datasets as an indicator of the retrieval difficulty, where a higher LID indicates greater difficulty.

Name	Dimension	Size	LID	Distance	Feature
DEEP	96	1 million 10 million 100 million 1 billion	3.9	Euclidean	Image
ByteECom1	128	1 million 10 million 100 million 700 million	5.6	Cosine	Graph
ByteECom2	128	1 million 10 million 100 million	10.7	Cosine	Graph
Uniform	32	1 million	5.1	Euclidean	Uniform Distribution
Gaussian	32	1 million with number of clusters from 1 to 100	5.6	Euclidean	Gaussian Distribution

TABLE IV: Dataset Summary

Machine and Measurement. All experimental indexing and evaluation tasks are conducted on a server equipped with an Intel(R) Xeon(R) Platinum 8336C CPU and 2TB of DRAM memory. This server features a total of 128 cores, all of which were utilized for both the construction and retrieval phases of our experiments using OpenMP for parallelism. This setup matches real-world application scenarios that require fast index construction and high throughput. The evaluation metrics employed in our study are as follows:

1. Construction Time: This metric measures the time consumption in seconds for constructing the ANNS index with a given parameter configuration.

2. Queries Per Second (QPS): This metric quantifies the throughput of the system, indicating the number of queries it can process per second.

3. Recall@10 for 10 Neighbors (Recall10@10): This metric measures the accuracy of the search, defined as the ratio of correctly identified nearest neighbors among the top 10 ground truth neighbors for queries.

Baselines, Implementations, and Parameter Settings. We benchmark Vista against leading graph-based indexes known for large-scale ANNS efficiency, including HNSW [40], HCNNG [43], Vamana [28], and LSHAPG [63], following guidelines from recent literature [9], [17], [61]. We carefully tune each baseline's construction parameters based on their research papers and documentation. For all benchmarks, we use their most optimized public implementations on large-scale datasets. ParlayANN [3] is used for Vamana and HCNNG, while an earlier HNSW version [1] and the original LSHAPG implementation [2] are used. We set the maximum edges to 128 and the search list size to 125 for HNSW and Vamana [28], configure HCNNG with 30 clusters and 1000 partitions [17], [41], and use the original LSHAPG setup [63]. Vista maintains a consistent configuration across experiments, except for Section V-E, which covers variations. Configurations are summarized in Table V.

Index	Construction Parameter Settings
HNSW	$efConstruction = 125, m = 64$
Vamana	$R = 128, L = 125, \alpha = 1.2$
HCNNG	$T = 50, Ls = 1000, s = 3$
LSHAPG	$K = 16, L = 2, T = 24, T' = 2T, p_\tau = 0.95$
Ours	$K = 50, R = 32, L = 100$

TABLE V: Summary of Construction Configuration

B. Evaluation on Search Performance

In this section, we report the search performance of all indexes based on Recall10@10 and QPS, excluding those unable to complete construction within 24 hours.

Achieving robust performance across datasets of varying sizes is critical for modern applications. To evaluate scalability, we analyze all indexes, including our proposed Vista, using the DEEP, ByteECom1, and ByteECom2 datasets at scales of 1M, 10M, 100M, and over 100M vectors. The results, shown in Figure 8 and Table VI, highlight Vista's performance across different scales. LSHAPG results are omitted where construction time exceeds 24 hours. We highlight ours Key observations below.

Superior Performance: Vista surpasses all baselines across the evaluated recall range on all datasets, confirming the efficiency of our index design and implementation. Among the evaluated baselines, Vamana is the second-best in efficiency and stability on average, followed by HCNNG, HNSW, and LSHAPG. Vista demonstrates impressive acceleration over Vamana, achieving 1.9 times faster performance at 0.98 recall on the DEEP-100M dataset and 1.5 times faster at 0.96 and 0.98 recall on the DEEP-1B dataset. Additionally, Vista achieves 4.0 times faster performance at 0.92 recall on the ByteECom1-100M dataset and 6.1 times faster performance at 0.80 recall on the ByteECom1-700M dataset. These results confirm the success of our index design in improving retrieval performance.

Improvement with Recall: The performance curve supports our statement regarding performance improvement with increasing recall. Across all evaluated real-world datasets of different scales, Vista consistently outperforms the baselines in terms of QPS, with a larger or at least comparable gap at higher recall levels.

For example, on the ByteECom1-100M dataset, acceleration improves from 1.4 to 2.0 for Vamana as the required recall increases from 0.88 to 0.92. On the ByteECom1-700M dataset, the acceleration over HCNNG increases from 2.3 to 8.5 as the required recall rises from 0.80 to 0.88. On the ByteECom2-10M dataset, the acceleration over Vamana increases from 1.5 to 3.4 as the required recall rises from 0.70 to 0.90. On DEEP datasets, the acceleration remains stable. This improvement is due to our optimization for hard-to-retrieve points with low k-NN connectivity. At higher recall levels, more challenging vectors need to be retrieved, causing baseline approaches to spend significantly more time searching for these points compared to Vista. These results confirm the effectiveness of Vista in handling points with low k-NN connectivity, thereby improving efficiency in the high-recall range.

Improvement with Scale: In addition to the superior efficiency of Vista across all evaluated scales, we observe greater or comparable improvements with increased scale, especially on industry datasets, indicating that Vista maintains or improves efficiency on larger datasets for the same recall performance. For example, compared to Vamana on ByteECom1, at a recall of 0.96, the acceleration for 10 million vectors is 2.7 times, which is greater than 1.5 times for 1 million vectors. On ByteECom2, the acceleration for 10 million vectors at 0.90 recall is 3.4 times, which is larger than 1.1 times at the same recall for 1 million vectors.

Compared to HCNNG, Vista shows greater or comparable improvement on larger datasets, with 4.1 times acceleration at 0.92 recall for ByteECom1 - 100M, which is larger than 1.6 times for 10 million vectors. Moreover, the acceleration for 700 million vectors at a recall of 0.88 is 8.5, significantly larger than the acceleration of 3.8 for 100 million vectors. On ByteECom2, the acceleration at 0.70 recall is 9.4 times for 100 million vectors, which is greater than 2.0 times for 10 million vectors. The acceleration at 0.90 recall for 10 million vectors is 3.4 times, compared to 1.3 times for 1 million vectors. These results indicate that with the same edge configuration, larger datasets may incur higher costs for retrieving imbalanced vectors. For smaller datasets, vectors can cover a relatively large local space with edges, thereby encompassing more vectors with low k-NN connectivity. This evaluation demonstrates the impressive scalability of Vista in efficiently handling large-scale datasets.

We observe that on the DEEP datasets, the improvement with increased scale and recall remains relatively stable. This may be because the DEEP dataset is inherently easier compared to the other two datasets (according to the LID), result-

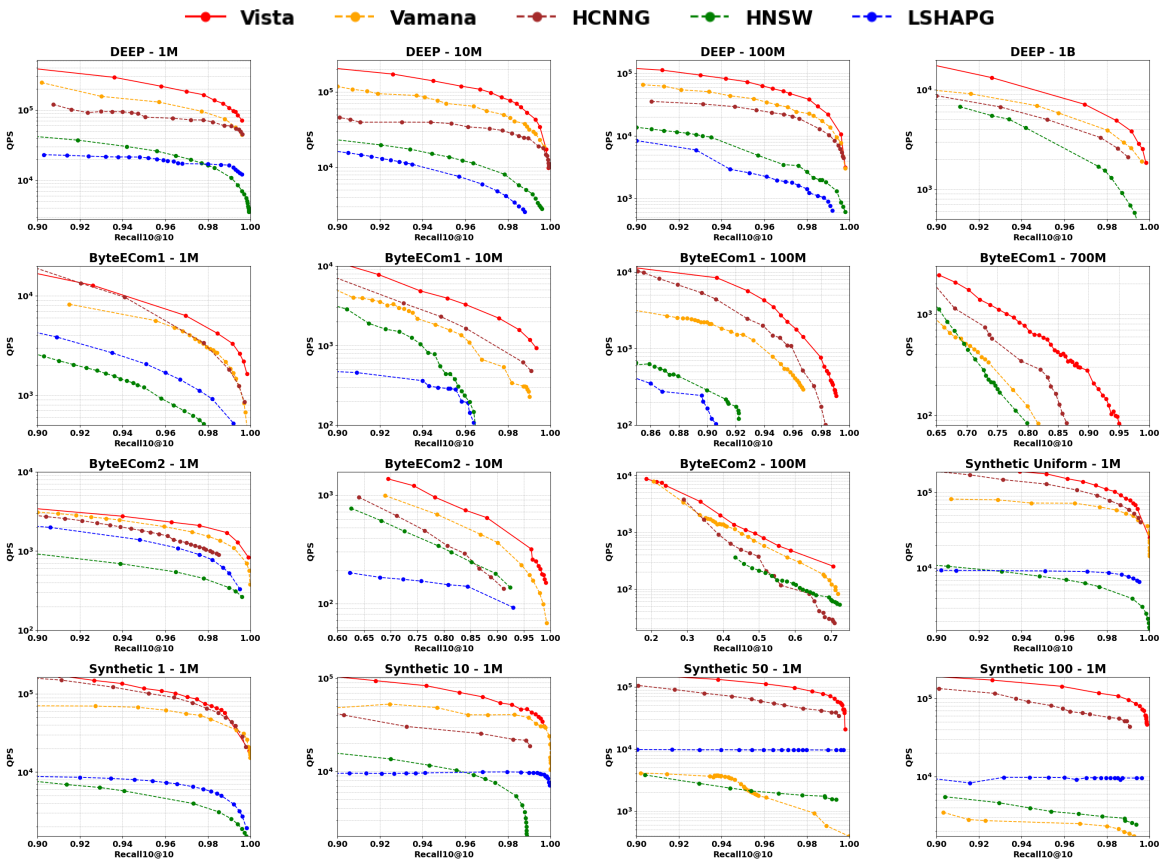


Fig. 8: Search performance with QPS-Recall curve on datasets in different scales.

ing in fewer increases in hard-to-retrieve vectors as scale and recall requirements rise. Consequently, this prevents a clear trend of improvement with scale or required recall across different settings. We also notice some GNN embedding datasets with high LID, such as ByteECom2, remain challenging even for state-of-the-art indexes. The recall performance for all indexes is lower than 0.70 when QPS is 300. This suggests that modern application datasets with indistinguishable high-dimensional vectors are still challenging and require more effective solutions.

C. Evaluation on Vector Distribution

To demonstrate the performance of Vista on various distributions, we evaluate Vista and baseline methods on synthetic datasets. Details on the k-NN distribution, visualizations, and further discussions on synthetic datasets can be found in our technical report. The search efficiency and the acceleration of Vista over baselines on synthetic datasets are reported in Figure 8 and Table VI. The results show that Vista consistently outperforms all baseline methods across all datasets. Vista demonstrates significant acceleration at high recall levels (i.e., > 0.90), achieving up to $2.6\times$ faster performance compared to the HCNNG index, which is often ranked as the second-best performer for synthetic datasets.

Vista's efficiency is particularly evident on datasets with multiple clusters, where its advantage over baselines increases as the number of clusters rises. This demonstrates Vista's effectiveness in handling complex distributions involving local clusters, a characteristic commonly seen in sequence-based models. In contrast, the baseline methods Vamana and HNSW show poor performance on datasets with many clusters, while Vista, HCNNG, and LSHAPG benefit from their use of space partitioning techniques such as clustering or hashing, resulting in more stable performance.

D. Evaluation on Construction Time

Here we report the construction time of all indexes over real-world datasets mentioned in the previous sections. The construction time is shown as a histogram in Figure 9. From the results, Vamana and HCNNG are the best-performing indexes across various datasets due to their effective algorithm design and efficient implementation in the ParlayANN library. Vista reports comparable construction time costs to Vamana and HCNNG, especially on datasets with scales up to 100 million. HNSW and LSHAPG are the least efficient solutions for all datasets due to their inefficient implementation in high-parallelism scenarios.

E. Discussion on Construction Parameters in Vista

Here we evaluate Vista's construction and search performance on the ByteECom2 dataset with 1 million vectors, focusing on the key parameter K. Evaluation and discussion

Index	DEEP-1M		DEEP-10M		DEEP-100M		DEEP-1B	
	Recall=0.96	Recall=0.98	Recall=0.96	Recall=0.98	Recall=0.96	Recall=0.98	Recall=0.96	Recall=0.98
Vamana	124491 (1.7×)	90060 (1.6×)	66529 (1.8×)	46833 (1.6×)	35545 (1.9×)	22995 (1.7×)	5628 (1.5×)	3943 (1.5×)
HCNNG	76898 (2.8×)	69726 (2.1×)	35044 (3.3×)	29258 (2.6×)	24785 (2.7×)	16656 (2.4×)	4477 (1.9×)	3037 (1.9×)
HNSW	24313 (8.9×)	16230 (9.0×)	12042 (9.7×)	7522 (10.2×)	4534 (14.7×)	2658 (15.0×)	2847 (3.0×)	1466 (3.9×)
LSHAPG	21301 (10.2×)	19320 (7.5×)	7108 (16.5×)	3973 (19.4×)	2789 (23.9×)	1420 (28.1×)	/	/
Vista	216211	145858	117289	77008	66544	39950	8617	5787
Index	ByteECom1-1M		ByteECom1-10M		ByteECom1-100M		ByteECom1-700M	
	Recall=0.96	Recall=0.98	Recall=0.92	Recall=0.96	Recall=0.88	Recall=0.92	Recall=0.80	Recall=0.88
Vamana	5189 (1.5×)	3020 (1.6×)	3557 (2.2×)	1229 (2.7×)	6716 (1.4×)	3171 (2.0×)	123 (6.1×)	< 80
HCNNG	6439 (1.2×)	3101 (1.6×)	4718 (1.6×)	1665 (2.0×)	2516 (3.8×)	1554 (4.1×)	324 (2.3×)	38 (8.5×)
HNSW	887 (8.9×)	462 (10.8×)	1701 (4.5×)	229 (14.3×)	434 (22.2×)	176 (36.0×)	82 (9.1×)	< 80
LSHAPG	1686 (4.7×)	985 (5.0×)	424 (18.1×)	195 (16.8×)	262 (36.8×)	26 (243.3×)	/	/
Vista	7905	4974	7664	3285	9652	6326	749	327
Index	ByteECom2-1M		ByteECom2-10M		ByteECom2-100M		Synthetic Uniform-1M	
	Recall=0.90	Recall=0.98	Recall=0.70	Recall=0.90	Recall=0.60	Recall=0.70	Recall=0.96	Recall=0.98
Vamana	3098 (1.1×)	1536 (1.2×)	950 (1.5×)	366 (3.4×)	321 (1.5×)	128 (2.2×)	71968 (2.2×)	61657 (1.8×)
HCNNG	2795 (1.3×)	1024 (1.8×)	690 (2.0×)	156 (3.4×)	100 (4.7×)	29 (9.4×)	116050 (1.3×)	80812 (1.4×)
HNSW	927 (3.8×)	435 (4.3×)	531 (2.6×)	184 (2.9×)	121 (3.9×)	65 (4.2×)	7076 (21.9×)	5290 (20.7×)
LSHAPG	2071 (1.7×)	819 (2.3×)	171 (8.1×)	110 (4.8×)	/	/	9099 (17.0×)	8682 (12.6×)
Vista	3542	1857	1388	532	472	276	154750	109290
Index	Synthetic 1-1M		Synthetic 10-1M		Synthetic 50-1M		Synthetic 100-1M	
	Recall=0.96	Recall=0.98	Recall=0.96	Recall=0.98	Recall=0.96	Recall=0.98	Recall=0.96	Recall=0.98
Vamana	62008 (1.7×)	49134 (1.5×)	41392 (1.7×)	40834 (1.3×)	1672 (67.2×)	1032 (85.8×)	2517 (56.2×)	2307 (49.1×)
HCNNG	94177 (1.7×)	64537 (1.3×)	26630 (2.6×)	22754 (2.3×)	57408 (2.0×)	44273 (2.0×)	73052 (1.9×)	57767 (2.0×)
HNSW	4678 (23.0×)	3460 (21.0×)	9788 (7.0×)	6316 (8.4×)	2018 (55.7×)	1778 (49.8×)	3442 (41.1×)	3036 (37.3×)
LSHAPG	8149 (20.7×)	5830 (12.5×)	9657 (8.7×)	9922 (5.4×)	9707 (13.5×)	9637 (9.2×)	9741 (16.4×)	9584 (11.8×)
Vista	181201	158078	68895	53100	112362	88507	141521	113193

TABLE VI: QPS performance of various indexes across different datasets and recall thresholds. Acceleration of Vista is highlighted in red.

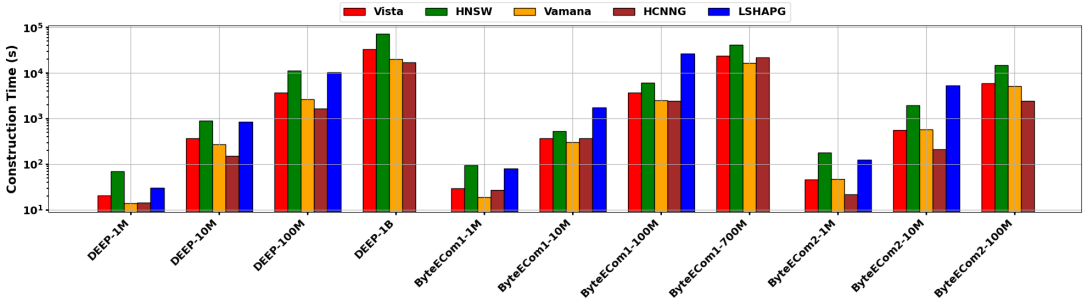


Fig. 9: Construction Time on Real-World Datasets.

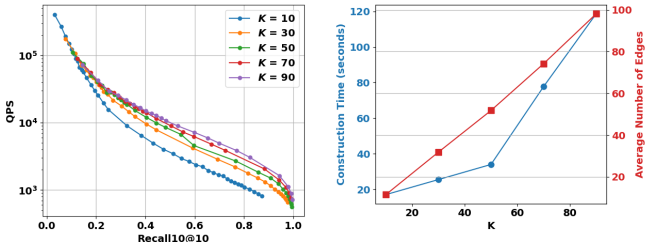


Fig. 10: Comparison on construction parameter setting on K.

on RPT parameters, i.e., R and L, can be found in our technical report. Our default setting is $K = 50$. We vary K and report the retrieval and construction results in Figure 10.

From the results with varying K, increasing K initially improves search performance in the high recall range, with a slight decrease in performance for the low recall range. Since industrial applications typically require high recall, using a larger K setting is generally recommended. However, the trade-off for larger K lies in increased time and memory costs, as the index construction requires more time and larger memory due to the increased number of graph edges.

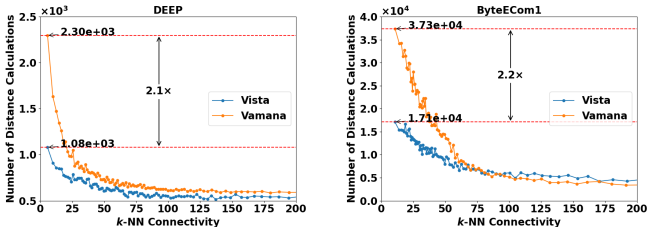


Fig. 11: Minimum number of distance calculations required to retrieve the ground truth vectors at different k-NN connectivity on two datasets. The ratio between the highest cost of Vamana and Vista is plotted.

F. Optimization for Vectors with Low k-NN Connectivity

To reveal the optimization results of Vista on vectors with low k-NN connectivity, we examine the search cost, indicated by the number of distance calculations, for vectors with different k-NN connectivity on Vista and the second-best baseline, Vamana, across two datasets, DEEP and ByteECom1, each with 1 million vectors. The results, shown in Figure 11, report the minimum number of distance calculations required to retrieve ground truth vectors and the ratio of search costs between the highest-cost vectors of the two methods. Vista demonstrates a clear reduction in the cost of retrieving vectors

with low k-NN connectivity, with search costs reduced by 2.1 and 2.2 times on two datasets, confirming its effectiveness in managing these vectors. This results in a slight increase in search cost for vectors with high k-NN connectivity. Since the cost of retrieving vectors with low k-NN connectivity is the bottleneck in ANNS systems (as indicated in Section III), Vista benefits from this trade-off.

VI. Conclusion

We introduce Vista, a dynamic index for ANNS that effectively addresses the challenges posed by imbalanced vector distributions. Vista improves the connectivity and retrieval efficiency of vectors with low k-NN connectivity, which are typically difficult to retrieve. Experimental results demonstrate that Vista outperforms state-of-the-art methods such as Vamana and HCNNG in efficiency. For instance, Vista achieved a 2.0 times acceleration at 0.92 recall on ByteECom1-100M over Vamana and a four-times acceleration over HCNNG. Additionally, Vista efficiently scales with large datasets to achieve high-recall responses and demonstrates an advantage in handling complex datasets with numerous local clusters, which is crucial for industrial applications, while maintaining construction times comparable to leading baselines.

References

[1] Hnswlib. https://github.com/nmslib/hnswlib, 2018.

[2] Lsh-apg. https://github.com/Jacyhust/LSH-APG, 2023.

[3] Parlayann. https://github.com/cmuparlay/ParlayANN, 2024.

[4] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[5] L. Amsaleg, O. Chelly, T. Furon, S. Girard, M. E. Houle, K.-i. Kawarabayashi, and M. Nett. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 29–38, 2015.

[6] F. André, A.-M. Kermarrec, and N. Le Scouarnec. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *42nd International Conference on Very Large Data Bases*, volume 9, page 12, 2016.

[7] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280. Citeseer, 1993.

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

[9] I. Azizi, K. Echihabi, and T. Palpanas. Elpis: Graph-based similarity search for scalable data science. *Proceedings of the VLDB Endowment*, 16(6):1548–1559, 2023.

[10] S. Berchtold, C. Böhm, and H.-P. Kriegal. The pyramid-technique: towards breaking the curse of dimensionality. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 142–153, 1998.

[11] B. Bratić, M. E. Houle, V. Kurbalija, V. Oria, and M. Radovanović. The influence of hubness on nn-descent. *International Journal on Artificial Intelligence Tools*, 28(06):1960002, 2019.

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[13] D. Caffagni, F. Cocchi, N. Moratelli, S. Sarto, M. Cornia, L. Baraldi, and R. Cucchiara. Wiki-llava: Hierarchical retrieval-augmented generation for multimodal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1818–1826, 2024.

[14] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

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

[16] Y. Ding, W. Fan, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li. A survey on rag meets llms: Towards retrieval-augmented large language models. *arXiv preprint arXiv:2405.06211*, 2024.

[17] M. Dobson, Z. Shen, G. E. Blelloch, L. Dhulipala, Y. Gu, H. V. Simhadri, and Y. Sun. Scaling graph-based anns algorithms to billion-size datasets: A comparative analysis. *arXiv preprint arXiv:2305.04359*, 2023.

[18] M. Dobson, Z. Shen, G. E. Blelloch, L. Dhulipala, Y. Gu, H. V. Simhadri, and Y. Sun. Scaling graph-based ANNS algorithms to billion-size datasets: A comparative analysis. *CoRR*, abs/2305.04359, 2023.

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

[20] A. Fernández, S. Garcia, F. Herrera, and N. V. Chawla. Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*, 61:863–905, 2018.

[21] C. Fu, C. Wang, and D. Cai. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4139–4150, 2021.

[22] C. Fu, C. Xiang, C. Wang, and D. Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*, 2017.

[23] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[24] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[25] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[26] P. Indyk and R. 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.

[27] J. W. Jaromczyk and G. T. Toussaint. Relative neighborhood graphs and their relatives. *Proceedings of the IEEE*, 80(9):1502–1517, 1992.

[28] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnawamy, and R. Kadekodi. Rand-nsg: Fast accurate billion-point nearest neighbor search on a single node. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 13771–13781. Curran Associates, Inc., 2019.

[29] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

[30] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, 2000.

[31] M. Li, Y.-G. Wang, P. Zhang, H. Wang, L. Fan, E. Li, and W. Wang. Deep learning for approximate nearest neighbour search: A survey and future directions. *IEEE Transactions on Knowledge and Data Engineering*, 35(9):8997–9018, 2022.

[32] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[33] P.-C. Lin and W.-L. Zhao. Graph based nearest neighbor search: Promises and failures. *arXiv preprint arXiv:1904.02077*, 2019.

[34] T. Liu, A. Moore, K. Yang, and A. Gray. An investigation of practical approximate nearest neighbor algorithms. *Advances in neural information processing systems*, 17, 2004.

[35] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. Gpt understands, too. *AI Open*, 2023.

[36] Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma. A survey of content-based image retrieval with high-level semantics. *Pattern recognition*, 40(1):262–282, 2007.

[37] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[38] K. Lu, M. Kudo, C. Xiao, and Y. 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.

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

[40] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

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

[42] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

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

[44] Y. Peng, B. Choi, T. N. Chan, J. Yang, and J. Xu. Efficient approximate nearest neighbor search in multi-dimensional databases. *Proceedings of the ACM on Management of Data*, 1(1):1–27, 2023.

[45] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/, 2014.

[46] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[47] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[48] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[49] M. Radovanović, A. Nanopoulos, and M. Ivanović. Nearest neighbors in high-dimensional data: The emergence and influence of hubs. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 865–872, 2009.

[50] M. Radovanovic, A. Nanopoulos, and M. Ivanovic. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research*, 11(sept):2487–2531, 2010.

[51] M. Reza, B. Ghahremani, and H. Naderi. A survey on nearest neighbor search methods. *International Journal of Computer Applications*, 95(25):39–52, 2014.

[52] A. Salemi and H. Zamani. Evaluating retrieval quality in retrieval-augmented generation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2395–2400, 2024.

[53] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*, pages 291–324. Springer, 2007.

[54] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[55] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[56] Y. Tian, X. Zhao, and X. Zhou. Db-lsh 2.0: Locality-sensitive hashing with query-based dynamic bucketing. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

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

[58] H. Wang, Z. Wang, W. Wang, Y. Xiao, Z. Zhao, and K. Yang. A note on graph-based nearest neighbor search. *arXiv preprint arXiv:2012.11083*, 2020.

[59] J. Wang, J. Wang, G. Zeng, R. Gan, S. Li, and B. Guo. Fast neighborhood graph search using cartesian concatenation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2128–2135, 2013.

[60] M. Wang, X. Xu, Q. Yue, and Y. Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631*, 2021.

[61] Z. Wang, P. Wang, T. Palpanas, and W. Wang. Graph-and tree-based indexes for high-dimensional vector similarity search: Analyses, comparisons, and future directions. *IEEE Data Eng. Bull.*, 46(3):3–21, 2023.

[62] J. Wei, B. Peng, X. Lee, and T. Palpanas. Det-lsh: A locality-sensitive hashing scheme with dynamic encoding tree for approximate nearest neighbor search. *arXiv preprint arXiv:2406.10938*, 2024.

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

APPENDIX

A. Discussion on hubs

Hubs, which are vectors with a disproportionately large number of out-edges in proximity ANNS graphs, significantly impact search efficiency. 'Good' hubs steer the search toward key regions, improving both efficiency and accuracy, while 'bad' hubs cause unnecessary computations by linking to irrelevant points. Therefore, effective hub management is vital in designing ANNS indices.

As shown in Table III, traditional ANNS graph methods like HNSW and Vamana apply a brute-force approach by suppressing all hubs using user-defined fixed bounds, based solely on edge count. This strategy neglects the routing benefits that some hubs may offer. Furthermore, as indicated in Equation 3, setting fixed upper bounds on graph edges can cause connectivity issues in high-dimensional, skewed datasets, leaving limited or no paths to some poorly connected vectors.

Unlike traditional methods, our approach, Vista, recognizes the importance of preserving certain hubs for effective routing, especially for vectors with low k-NN connectivity. By implementing an adaptive strategy—dynamic edge exchange—we retain beneficial 'good' hubs for routing while reducing unnecessary computation on irrelevant points using the distance-based RNG strategy on edges. This balance enhances search efficiency while maintaining the advantages of key hubs. The following experiments confirm these findings.

To evaluate the impact of hub control strategies on index structure and retrieval performance, we experiment with the two described rules and present the results. All strategies use the same initialized approximate k-NN graph of three real-world datasets with 1 million vectors, following the same candidate generation and edge selection processes. The only difference is in how each strategy manages the final size of out-edges for each vector to control hubs. All other configurations are as specified in Section V-A. We compare the rigid and flexible strategies in terms of generated edges and search efficiency, demonstrating that our proposed hub control method is more efficient.

Strategy 1. Rigid edge size control: If the number of out-edges exceeds a fixed upper bound, only the closest edges are preserved. The upper bound is set to K, i.e. the approximate number of neighbors computed during initialization.

Strategy 2. Flexible edge size control: We follow the dynamic edge exchange mechanism proposed in Algorithm 6.

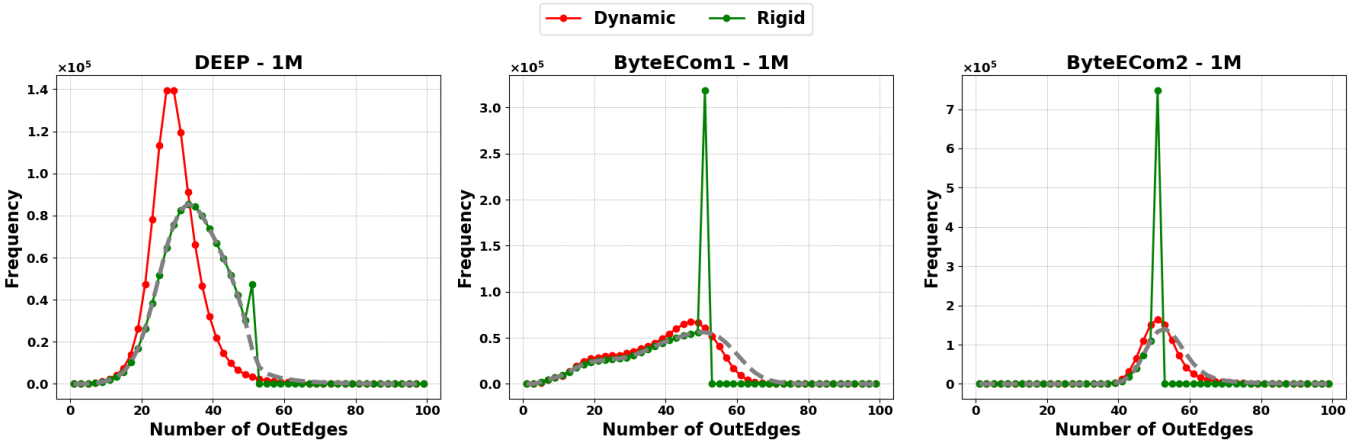


Fig. 12: Frequency of out-degree on various datasets. The grey line represents no hubness control, meaning no edges are pruned.

Experiment 1: Evaluation on Edge Structure

Figure 12 shows the out-degree distribution of two strategies across different datasets, with the grey dotted line representing the original edge distribution without pruning. The traditional rigid method clips edges once the out-degree exceeds a set threshold ($K = 50$), resulting in zero frequency beyond 50 (green line) and many vectors reaching the maximum out-degree.

In contrast, our flexible strategy adjusts out-degree control based on vector and edge distribution, where vectors with higher out-degrees are more likely to be pruned (as described in Algorithm 6, edge exchange occurs from higher to lower out-degree vectors). This approach reduces, but does not eliminate, vectors with high out-degrees, resulting in a larger proportion of vectors with moderate edge counts. It also retains key edges for vectors with low k-NN connectivity, while the traditional method removes them, negatively impacting accuracy for poorly connected vectors.

Experiment 2: Evaluation on Retrieval Performance

In this experiment, we compare the search efficiency of two strategies on three datasets in terms of recall and speed, as shown in Figure 13.

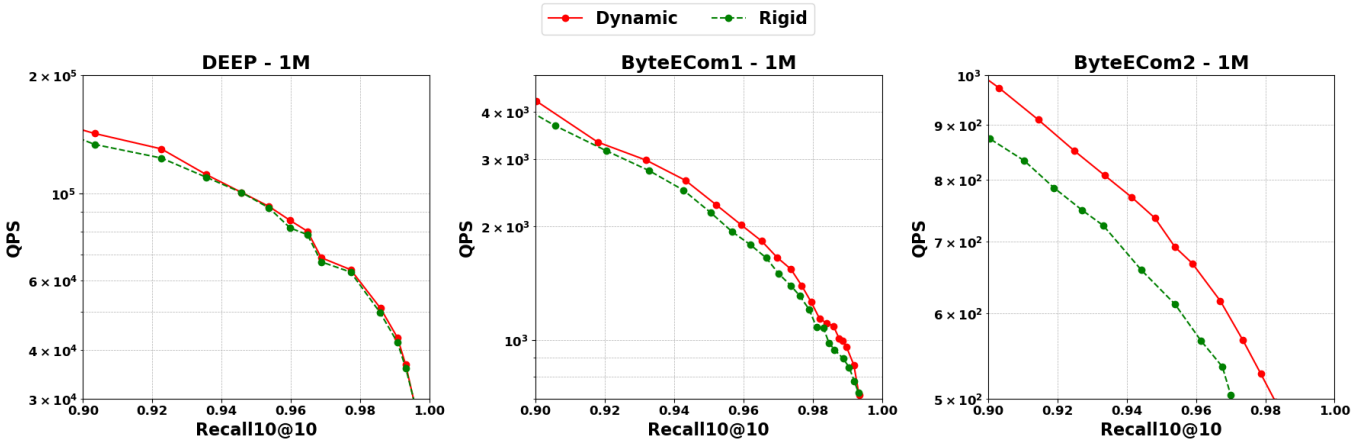


Fig. 13: Search performance comparison between two strategies. Our proposed strategy is more efficient on all datasets in high recall range.

Our dynamic strategy consistently achieves better or comparable speed and accuracy across high recall values. This improvement is especially pronounced in the two GNN industrial datasets, where the rigid hub control strategy removes many edges from high out-degree vectors, increasing retrieval costs for vectors with low k-NN connectivity.

To summarize our discussion on hub control, hubs are vital for the search efficiency of graph-based ANNS indices. Achieving a balance between effective routing (connecting to relevant vectors) and minimizing unnecessary connections is key during index construction. Traditional methods simply cap the number of edges, pruning all beyond a fixed out-degree. In contrast, our approach adapts to vector distribution, selectively reducing edges to unrelated vectors while preserving crucial connections for vectors with low k-NN connectivity. Our evaluations across multiple datasets confirm its effectiveness.

B. Discussion on Compensation for Low Connectivity Points

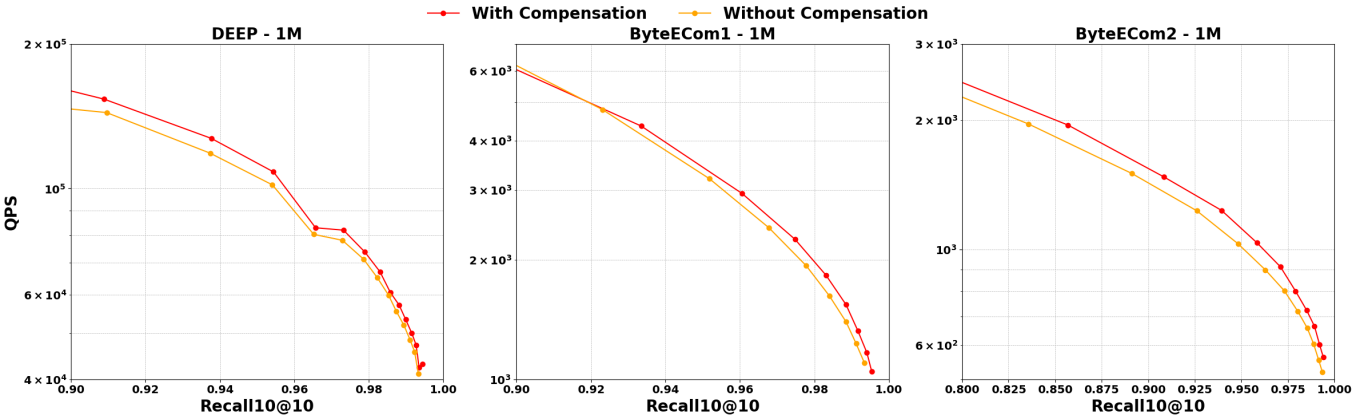


Fig. 14: Comparison between compensation strategies on vectors with low k-NN connectivity.

Balancing the distribution of constructed edges significantly affects edge allocation across vectors with different distributions. Adding more edges to sparse vectors with low k-NN connectivity can direct the search process towards these regions, potentially leading to inefficiencies if the query is not located in a sparse area. However, as discussed in Section III, vectors with low k-NN connectivity in sparse regions often become bottlenecks for traditional graph indexes when striving for high-recall performance. Therefore, constructing additional edges to these hard-to-find vectors is essential.

The key is to ensure appropriate compensation in the graph for vectors with low k-NN connectivity. In our design, this compensation is introduced through the number of edge candidates in the acquisition process, as defined in Equation 4: $C_i = K + (K - \min(K, p_i))$. To assess the effectiveness of this approach and its prevention of excessive edge creation for sparse vectors, we provide both theoretical and experimental analyses.

a. Discussion on Theoretical Bound and Estimation

In this section, we aim to demonstrate that while our design compensates for constructing in-edges for vectors with low k-NN connectivity, the upper bound of the in-degree for vectors with high k-NN connectivity ($> K$) is greater than that for

vectors with low k-NN connectivity ($< K$). We follow the notations defined in Table II and start the analysis with Equation 1 for in-edges of vectors in traditional graph indexes:

$$\begin{aligned}\mathcal{I}(v_i) = &\mathcal{P}(v_i, n, (\mathcal{S}(v_i, \mathcal{N}(v_i, m))))\\ &\cup \mathcal{P}(v_i, n, \{v_j \mid v_i \in \mathcal{S}(v_j, \mathcal{N}(v_j, m))\})\end{aligned}$$

We employ an element-wise upper bound on the number of edge candidates along with distribution-based edge pruning. For a given vector v_i, the upper bound of edge candidates is defined as $m_i = \alpha \cdot C_i = \alpha \cdot (K + (K - \min(K, p_i)))$. The edge pruning process, represented by $\mathcal{T}(v_i, \mathcal{X})$, selects a set of vectors from $\mathcal{X}$ such that v_i appears among the outgoing edges resulting from the dynamic edge exchange of vectors in $\mathcal{X}$. Consequently, the in-edges of vector v_i in the Vista graph, denoted by $\mathcal{I}^{'}(v_i)$, are:

$$\begin{aligned}\mathcal{I}^{'}(v_i) = &\mathcal{T}(v_i, (\mathcal{S}(v_i, \mathcal{N}(v_i, \alpha \cdot C_i))))\\ &\cup \mathcal{T}(v_i, \{v_j \mid v_i \in \mathcal{S}(v_j, \mathcal{N}(v_j, \alpha \cdot C_i))\})\end{aligned}$$

Since the edge selection process with RNG is highly dependent on vector distribution, we begin with a simplified scenario where no edge pruning occurs during the edge selection. In this case, we present:

1. The in-edges for vectors with low k-NN connectivity ($< K$) are bounded by a value solely related to the scalar parameter K.

2. The upper bound of in-edges for vectors with high k-NN connectivity ($> K$) is greater than that of vectors with low k-NN connectivity.

In this case, we have $\mathcal{S}(v_i, \mathcal{X}) = \mathcal{X}$. Also, from the definition of α in Equation 5, $\alpha = K/P = 1.0$. Thus we have:

$$\begin{aligned}\mathcal{I}^{'}(v_i) = &\mathcal{T}(v_i, \mathcal{N}(v_i, C_i))\\ &\cup \mathcal{T}(v_i, \{v_j \mid v_i \in \mathcal{N}(v_j, C_i)\})\\ \subseteq &\mathcal{N}(v_i, C_i) \cup \{v_j \mid v_i \in \mathcal{N}(v_j, C_i)\}\end{aligned}$$

From the above equation, the first set represents the neighboring vectors of v_i, while the second set consists of vectors v_j that consider v_i as an edge candidate. Without any assumptions regarding the distribution of v_j, we approximate the second set of v_i as $\mathcal{M}(v_i, K)$. We have already established the k-NN relationship with $k = K$, where $|\mathcal{M}(v_i, K)| = p_i$. Therefore, we have:

$$\begin{aligned}\mathcal{I}^{'}(v_i) &\subseteq \mathcal{N}(v_i, C_i) \cup \{v_j \mid v_i \in \mathcal{N}(v_j, C_i)\}\\ &\approx \mathcal{N}(v_i, C_i) \cup \mathcal{M}(v_i, K)\end{aligned}$$

Consider the size of in-edges for v_i, we have:

$$\begin{aligned}|\mathcal{I}^{'}(v_i)| &\leq |\mathcal{N}(v_i, C_i)| + |\mathcal{M}(v_i, K)|\\ &= 2 \cdot K - \min(K, p_i) + p_i\end{aligned}$$

From the above expression, for vectors with low k-NN connectivity ($p_i < K$), the upper bound for in-edges is $2 \cdot K$. In contrast, for vectors with high k-NN connectivity, the upper bound is $K + p_i > 2 \cdot K$.

In the second case, when considering the RNG strategy for edge selection, we have $\alpha > 1.0$, and $\alpha \cdot C_i$ edge candidates are generated for each vector. Given that the edge selection process is significantly influenced by the vector distribution, we provide an estimation of the upper bound of in-degree based on approximate assumptions for vectors in different regions.

Assumption 1: The RNG edge selection process selects edges for vectors in a linear manner, meaning that the number of resulting edges is $\frac{1}{\alpha}$ of the total number of edge candidates.

Assumption 2: The k-NN connectivity remains stable as k increases, i.e., $|\mathcal{M}(v_i, \alpha \cdot K)| \approx \alpha| \cdot \mathcal{M}(v_i, K)|$.

Both assumptions are highly dependent on the value of α. In our applications, α is relatively small, i.e., < 3.0, which makes these assumptions approximately valid. Based on these assumptions, we have:

1. The in-edges for vectors with low k-NN connectivity ($< K$) are bounded by a value related to scalar parameters K and α.

2. The estimated bound on the in-degree of vectors with high k-NN connectivity is higher than that of vectors with low k-NN connectivity.

Consider above assumptions in $\mathcal{I}^{'}(v_i)$, we have:

$$\begin{aligned}|\mathcal{I}^{'}(v_i)| \leq & |\mathcal{T}(v_i, (\mathcal{S}(v_i, \mathcal{N}(v_i, \alpha \cdot (C_i)))))| \\ & + |\mathcal{T}(v_i, \{v_j \mid v_i \in \mathcal{S}(v_j, \mathcal{N}(v_j, \alpha \cdot (C_i)))\})| \\ \leq & |\mathcal{S}(v_i, \mathcal{N}(v_i, \alpha \cdot (C_i)))| \\ & + |\{v_j \mid v_i \in \mathcal{S}(v_j, \mathcal{N}(v_j, \alpha \cdot (C_i)))\}| \\ \approx & \alpha \cdot C_i + \alpha \cdot p_i = \alpha \cdot (C_i + p_i) \\ = & \alpha \cdot (2 \cdot K - \min(K, p_i) + p_i))\end{aligned}$$

From the above expression, for vectors with low k-NN connectivity ($p_i < K$), the upper bound for in-edges is $2 \cdot \alpha \cdot K$. For vectors with high k-NN connectivity, the upper bound is $\alpha \cdot (K + p_i) > 2 \cdot \alpha \cdot K$, which is larger than the approximate bound for vectors with low k-NN connectivity.

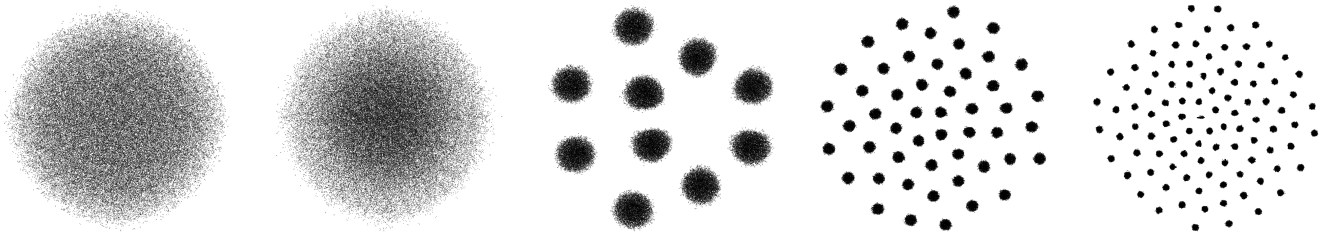
b. Evaluation of Compensations for Vectors with low k-NN Connectivity

In addition to the theoretical analysis, we investigate the degree of compensation for vectors with low k-NN connectivity through experiments. In the Vista construction process, additional neighbors are identified for these vectors to generate more in-edges from their neighbors. This is managed by adjusting the parameter for the number of neighbor candidates during edge selection. In this evaluation, we compare the effects of enabling or disabling compensation on edge candidates during edge construction. In the setting of disabling compensation, we use the same setting on the number of edge candidates for all vectors $C_i = K$. In Figure 14, we present the search performance of two strategies on three million scale datasets, it can be observed from the result that strategy with compensation for low k-NN connectivity vectors outperforms the competitor at all evaluated high-recall range, confirming its effectiveness on efficiency.

To summarize, we address the concern of additional edge construction for low-connectivity vectors in sparse regions and its impact on search efficiency through both theoretical analysis of our design and evaluations on various datasets. Our analysis demonstrates that, while compensation is made for low k-NN connectivity vectors, it remains controlled by (approximate) bounds set by scalar parameters, and the upper bound for high k-NN connectivity vectors is larger. The evaluation results show that strategies incorporating compensation for low k-NN connectivity vectors are effective across real datasets, leading to improved search efficiency. Therefore, concerns regarding the compensation are unlikely to hold true.

C. Stree-test on Skewness with Syntehtic Datasets

We use synthetic datasets that follow either uniform or Gaussian distributions. For the Synthetic Uniform dataset, each dimension is independently generated from a uniform distribution. For datasets with Gaussian distributions, we vary the number of local clusters from 1 to 100 to introduce different levels of skewness. The center of each cluster is defined based on the binary representation of the cluster ID; for example, the first cluster is centered at $[0, \ldots, 1]$, and the second cluster is centered at $[0, \ldots, 1, 0]$. We ensure that all clusters are of equal size and that the data points within each cluster follow a Gaussian distribution with a variance of 1.0 in each dimension.



(a) Synthetic Uniform (b) Synthetic 1 (c) Synthetic 10 (d) Synthetic 50 (e) Synthetic 100

Fig. 15: Visualization of Synthetic datasets with t-SNE.

We use t-SNE to visualize the synthetic datasets in a 2-D space, as shown in Figure 15. The figure clearly illustrates that the datasets exhibit either a uniform or Gaussian distribution, with those containing multiple clusters displaying well-defined, uniformly distributed clusters of equal size.

We analyze the distribution of k-NN connectivity across various synthetic datasets and visualize the results in Figure 16. The results clearly indicate the differences in skewness among the synthetic datasets: datasets with uniform distributions have

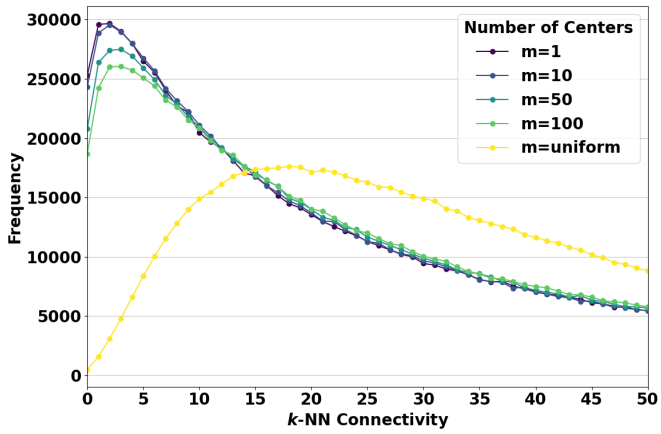


Fig. 16: Distribution of k-NN connectivity of various datasets.

a higher proportion of vectors with high k-NN connectivity, while datasets with Gaussian distributions contain more vectors with low k-NN connectivity. When comparing datasets with different numbers of clusters, those with more clusters exhibit higher k-NN connectivity for a larger number of vectors, while fewer vectors have low k-NN connectivity. In summary, these datasets display distinct k-NN connectivity distributions, making them suitable for stress-testing purposes.

Dataset	Recall					Dataset	Recall				
	0.90	0.92	0.94	0.96	0.98		0.90	0.92	0.94	0.96	0.98
Acceleration over Vamana						**Acceleration over HCNNG**					
Synthetic Uniform	3.13	2.83	2.51	2.15	1.77	Synthetic Uniform	1.37	1.39	1.34	1.33	1.35
Synthetic 1	2.57	2.26	1.98	1.74	1.48	Synthetic 1	1.14	1.13	1.16	1.14	1.13
Synthetic 10	3.51	2.45	2.26	2.11	1.49	Synthetic 10	4.07	3.68	3.83	3.29	2.67
Synthetic 50	73.26	64.95	55.06	96.81	97.50	Synthetic 50	2.85	2.82	2.73	2.82	2.27
Synthetic 100	97.11	114.00	97.27	75.66	57.52	Synthetic 100	2.62	2.58	2.66	2.61	2.30
Acceleration over HNSW						**Acceleration over LSH-APG**					
Synthetic Uniform	23.61	23.47	22.25	21.87	20.66	Synthetic Uniform	27.51	24.48	20.29	17.01	12.59
Synthetic 1	23.91	23.64	23.42	23.05	20.99	Synthetic 1	20.69	18.44	16.59	14.67	12.46
Synthetic 10	10.81	9.07	9.36	8.94	9.61	Synthetic 10	17.77	13.36	11.67	8.90	6.11
Synthetic 50	75.94	79.29	81.86	80.18	56.59	Synthetic 50	30.78	25.79	20.76	16.77	10.44
Synthetic 100	62.79	63.42	61.90	55.32	43.72	Synthetic 100	38.08	36.12	26.19	19.69	13.85

TABLE VII: Acceleration results over different methods across synthetic datasets

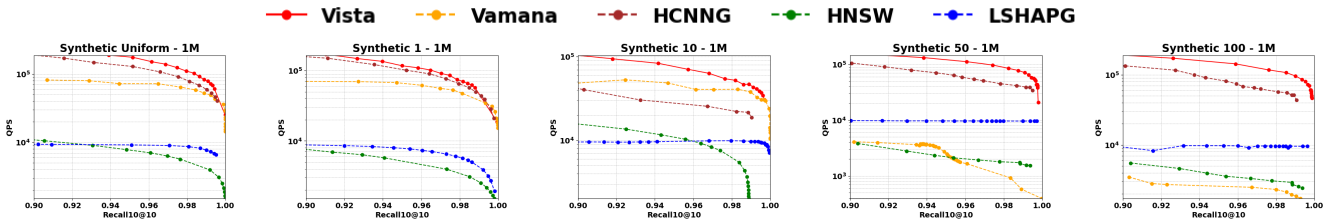


Fig. 17: Evaluation of search performnace on synthetic datasets.

The results of the search performance evaluation of Vista and four advanced baselines are presented in Figure 17. Vista shows a clear advantage over all baseline methods across all datasets. For a clearer comparison, we present the acceleration of Vista over the baselines at different recall thresholds in Table VII. From the results, we conclude several important points:

1. Superior Performance: Vista shows a significant advantage over all baselines, particularly with impressive acceleration at high recall levels (i.e., > 0.90). In most cases, HCNNG is the second-best approach, and Vista achieves acceleration ranging from 1.33× to 4.07× across different recall levels or datasets. For other baselines, Vista achieves even higher acceleration, often by tens of times. These results demonstrate that Vista is both efficient and robust across datasets with varying distributions.

2. Advantages of Vista on Different Datasets: Comparing performance across different datasets, we observe that for all ANNS indexes, their efficiency, measured as QPS at a certain recall, decreases as the number of clusters increases. This suggests that datasets with more local clusters are more challenging for modern graph-based ANNS approaches. Notably, Vista excels in handling such datasets with multiple clusters, as its acceleration over the baselines grows with the increasing number of clusters. This confirms Vista's effectiveness in addressing the ANNS task for datasets with complex distributions involving local clusters, which is crucial since many sequence-based models generate vector embeddings with distinct local clusters, as we introduced in Section I.

3. Baselines on Datasets with Multiple Clusters: We observe that the baseline methods Vamana and HNSW perform poorly on datasets with multiple clusters (e.g., 50 and 100 clusters), whereas the performance of the other three indexes is more stable. One possible explanation is that Vista, HCNNG, and LSH-APG incorporate a space partitioning component in their indexing process, which divides the feature space into smaller regions using methods like LSH or Random Projection Trees for further graph search. This space-partitioning structure helps manage local clusters more effectively, thereby improving efficiency.

D. Discussion on vector skewness on various types of indexes

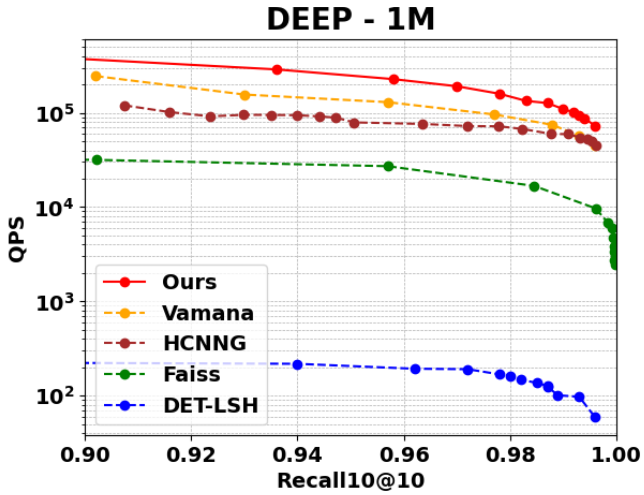


Fig. 18: Search performance comparison between various types of indexes on DEEP dataset.

Firstly, we focus our research on graph-based approaches, as they are the leading solutions in industrial-grade applications due to their high efficiency and accuracy. This observation is supported by various evaluation benchmarks conducted on datasets ranging from millions to billions in scale [8], [9], [32], [58], [60], which we also validate in our evaluation below. As suggested by ANN-Benchmark [8], tree, hashing, and graph methods are recommended types of indexes for large-scale ANNS. Therefore, we compare graph-based methods with the state-of-the-art tree-based method (inverted index) IVFPQFS [6] in Faiss library [29] and the hashing-based method DET-LSH [62] on million-scale datasets, presenting their retrieval performance in Figure 18. From the evaluation results on the DEEP-1M dataset, it is evident that graph-based methods, including Vista, Vamana, and HCNNG, deliver search speeds magnitudes faster than tree-based and hashing-based methods, achieving substantial acceleration in the high-recall range, making them more suitable for industrial-grade applications at ByteDance.

Secondly, we explore the impact of dataset imbalance on the retrieval performance of various ANNS indexes, including the graph-based method Vamana, the tree-based method Faiss, and the hashing-based method DET-LSH. To begin, we examine the distribution of search costs for database vectors, represented by the number of distance calculations required by each ANNS index to identify the correct ground truth vectors. The frequency distribution of search costs per vector is illustrated in Figure 19.

From the results, we observe that the tree-based method Faiss and the hashing-based method DET-LSH exhibit a more concentrated distribution of search costs, while the graph-based method Vamana demonstrates a much wider range of search costs. This indicates that searches using the graph-based method can vary significantly in computation cost, depending heavily on the distribution of the query, ground truth, and the underlying index structure. Therefore, it is both meaningful and beneficial to explore opportunities for improving the efficiency of searching for extremely hard-to-find vectors in graph-based indexes. For the other indexes, since their search costs are relatively consistent across queries, optimizing query efficiency based on cost distribution is less significant and beneficial compared to graph-based indexes.

Finally, we also examine the distribution of search costs for tree-based and hashing-based indexes with respect to the k-NN connectivity of vectors, which serves as an indicator of search difficulty identified in this work. We plot the search cost of ground truth vectors against their k-NN connectivity in Figure 20 on the DEEP-1M dataset. For clear visualization, each point in the figure represents the center of 1,000 nearby points.

From the results, the graph-based index shows a distinct trend between k-NN connectivity and search cost: as k-NN connectivity increases, the search cost decreases. For the tree-based index Faiss, search costs are notably high for vectors with very low k-NN connectivity (e.g., < 30). For other vectors, there is no clear relationship. This occurs because vectors with low k-NN connectivity tend to be distributed in sparse regions of the feature space, making it challenging for the space partition algorithm to allocate these vectors into appropriate clusters. For the hashing-based method DET-LSH, there is no

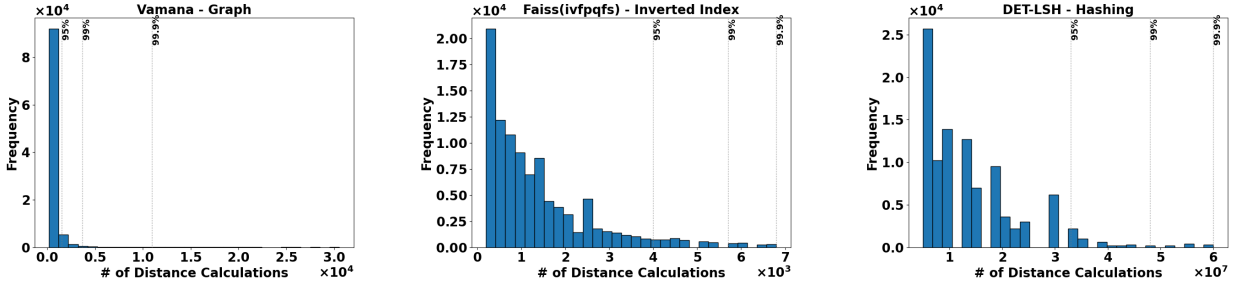


Fig. 19: Distribution of search cost, i.e. number of distance calculations of various indexes on DEEP dataset.

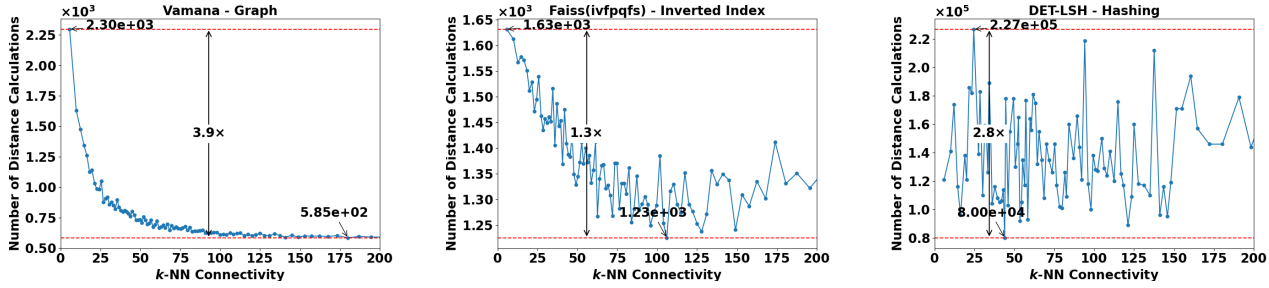


Fig. 20: Distribution of search cost with k-NN connectivity on various types of indexes.

evident relationship between k-NN connectivity and search cost. Consequently, both the tree-based and hashing-based methods are not particularly sensitive to k-NN connectivity in the retrieval task.

In summary, we conducted a series of experimental evaluations on various types of indexes to analyze the impact of vector skewness on the search efficiency of different index types. We begin by emphasizing our focus on graph-based methods, as they have proven to be the most efficient solutions for large-scale industrial ANNS tasks, meeting the requirements of both high accuracy and speed. Our evaluation confirms the superiority of graph indexes, achieving significant acceleration compared to other methods. Furthermore, we investigate the distribution of search costs across different index types to identify whether some vectors incur disproportionately higher costs than others. The results show that search cost distributions in tree-based and hashing indexes are relatively stable and concentrated, indicating that only a few vectors have unusually high search costs, limiting the potential for optimization targeting skewed vectors. Lastly, we examine the relationship between search cost distributions and the skewness indicator, k-NN connectivity, across different indexes. Our findings reveal a clear negative correlation between search costs and k-NN connectivity in graph-based indexes, whereas other indexes show no such correlation. Overall, tree-based and hashing indexes are more robust to vector skewness, making them less suitable for optimization based on data skewness compared to graph-based methods.

E. Discussion on construction parameters

The parameters that require user definition, as outlined in Algorithm 1 and Table V, are three parameters: the maximum leaf size in the random projection tree (RPT) L, the number of RPTs R, and the number of nearest neighbors for initialization K. Among these, K is the most critical parameter for Vista's index construction, as it directly influences the graph edge construction process. Evaluation and discussion on K are provided in Section V-E. Below, we present the discussion and evaluation of the remaining parameters, R and L on their impact on construction and retrieval performance. R and L are parameters for RPTs construction, which are used to generate the initialization of the approximate k-NN graph for further Vista graph construction and query retrieval.

1. Evaluation on R:

From the results shown in Figure 21, increasing R slightly enhances search performance, as indicated by the upward shift in QPS. Higher values of R (e.g., 64 and 96) consistently achieve better QPS at high recall levels compared to lower values (e.g., 16 and 32), suggesting that a larger R allows for more effective retrieval by expanding the pool of search candidates. The figure also illustrates the trade-offs associated with increasing R. As R increases, both construction time and the average number of edges in the index rise. In particular, construction time spikes sharply between $R = 40$ and $R = 60$, while the average number of edges peaks at $R = 60$. Beyond this point, both metrics plateau, indicating diminishing returns in performance relative to the increased costs. These results emphasize that while larger R values improve search efficiency, they also demand more computational resources and memory, necessitating a careful balance based on specific application requirements.

2. Evaluation on L:

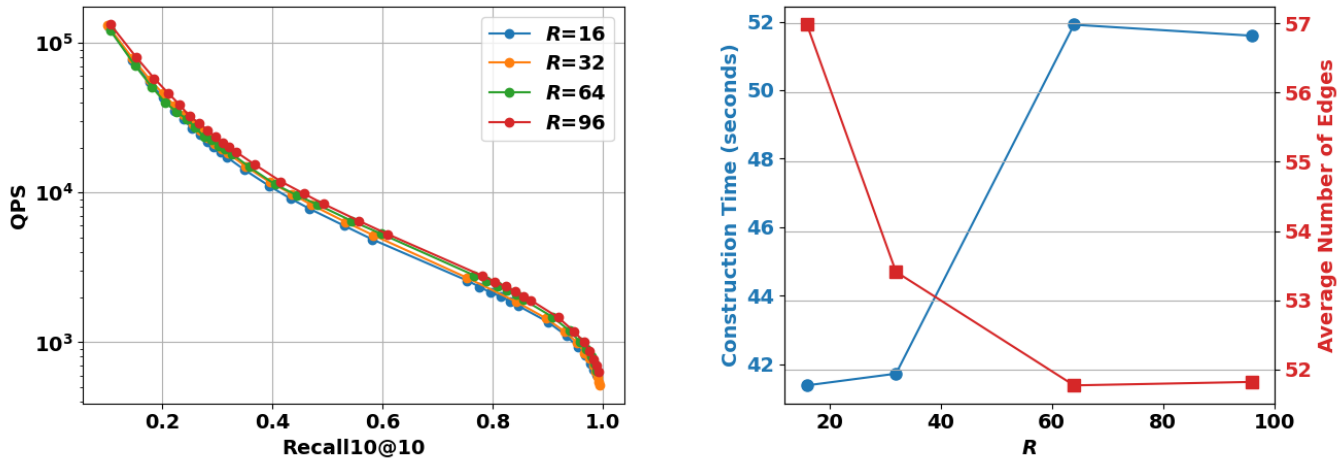


Fig. 21: Comparison on Construction Parameter Settings on R.

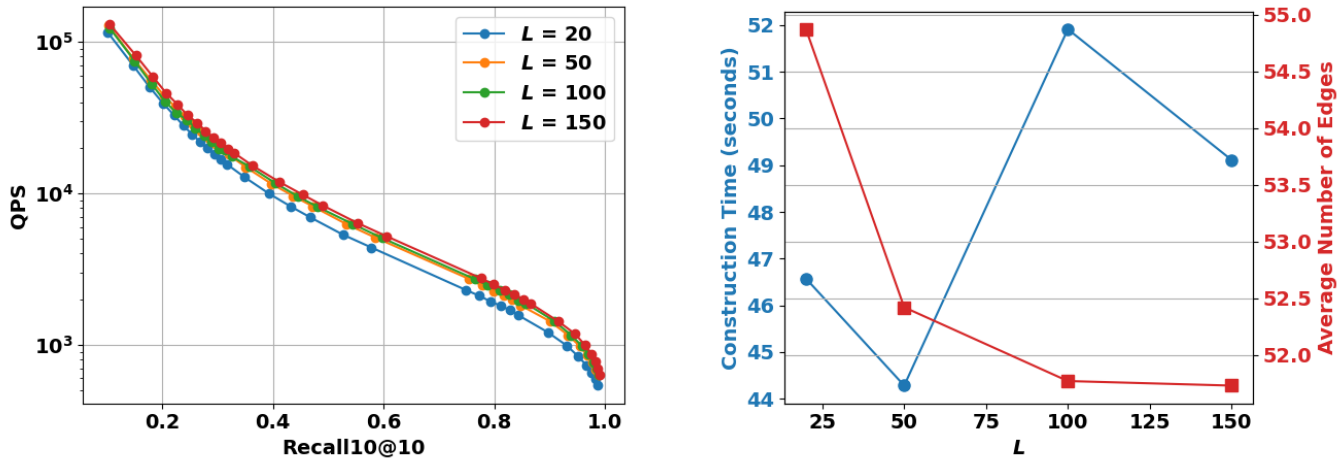


Fig. 22: Comparison on Construction Parameter Settings on L.

As shown in Figure 22, increasing L, which controls the leaf size in the RPT, generally improves search performance, particularly in the high recall range. In Figure 22, higher values of L (e.g., 100 and 150) achieve better QPS at high recall levels compared to lower values (e.g., 20 and 50), indicating that larger leaf sizes enhance the retrieval process by accommodating more candidates within each search iteration. The figure also shows that the construction time remains relatively consistent across different values of L, with no clear trend observed. This suggests that construction costs are fairly stable regardless of the choice of L. Meanwhile, the average number of edges slightly decreases as L increases, leading to a more compact index structure. These results suggest that larger L values can enhance search efficiency without significantly impacting construction time, allowing for a balanced performance depending on application needs.

F. Notes on Implementation

The implementation of the ANN index is crucial for its performance in construction as well as the retrieval phase. We record some of our important implementation techniques which found to be beneficial to graph-based indexes. We follow the non-lock pattern introduced in ParlayANN [18] for RPT construction, edge candidate acquisition, and edge selection.